UNIVERSIDAD SAN FRANCISCO DE QUITO

Una Infraestructura Genérica para la Implementación de Aplicaciones Distribuidas en Internet

Pablo Bustamante Sandoval

Tesis de Grado presentada como requisito para la obtención del título de

Ingeniero en Sistemas

Quito
Octubre 2006

UNIVERSIDAD SAN FRANCISCO DE QUITO

Colegio Politécnico

HOJA DE APROBACIÓN DE TESIS

Una Infraestructura Genérica para la Implementación de Aplicaciones Distribuidas en Internet

Pablo Bustamante Sandoval

Enrique Vinicio Carrera, DSc ....
Director de Tesis (firma)

Fausto Pasmay, MS ....
Miembro del Comité de Tesis (firma)

Julio Arauz, PhD ....
Miembro del Comité de Tesis (firma)

Santiago Valencia, MS ....
Miembro del Comité de Tesis (firma)

Ramiro Delgado, MS ....
Miembro del Comité de Tesis (firma)

Fernando Romo, MS ....
Decano del Colegio Politecnico (firma)

Quito

Octubre 2006



Deseo dedicar esta tesis a mis queridos padres y hermanas

Agradecimientos

Agradezco a mi tutor Vinicio Carrera por la paciencia y la inmensa ayuda para terminar esta tesis, la cual sin su aporte no sería lo que es hoy día.

Gracias.

Abstract

This dissertation proposes a generic infrastructure for the developing of parallel and distributed applications on the Web. This infrastructure is oriented to allow that every single host in the Internet can participate on the execution of distributed applications using a very simple configuration with rigid guarantees of security. Our proposal is based on the use of the World Wide Web protocols and Java applets, exclusively. Thus, users willing to participate only require a conventional Web browser with the Java Runtime Environment enabled. This work differs from previous proposals in its simplicity and improved performance. In addition, our proposal includes a detailed performance evaluation of real parallel applications.

Resumen

Esta tesis propone una infraestructura genérica para el desarrollo de aplicaciones distribuidas y paralelas sobre la Web. Esta infraestructura está orientada a permitir que todos los usuarios de Internet sean capaces de participar en la ejecución de aplicaciones distribuidas, usando una configuración simple pero con claras garantías de seguridad. Nuestra propuesta se basa en el uso de protocolos disponibles en la *World Wide Web* y *applets* Java exclusivamente. Por ende, el usuario que desee participar de esta infraestructura sólo requerirá de un navegador convencional que tenga habilitado su JRE (*Java Runtime Environment*) correspondiente. Nuestra propuesta difiere de las anteriores por su simplicidad y mejor desempeño. Adicionalmente, nuestro trabajo incluye una evaluación detallada de la ejecución de aplicaciones reales sobre nuestra infraestructura.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

El Internet que conocemos ha sido producto de un rápido crecimiento durante los últimos años, inter–conectado a miles de millones de computadoras alrededor del mundo. La mayoría de estos computadores, en especial los computadores de hogar, se encuentran funcionando muy por debajo de sus capacidades reales, sobre todo si tomamos en cuenta que las capacidades de procesamiento en estas máquinas ha crecido exponencialmente. Teniendo en cuenta esta enorme capacidad de procesamiento subutilizada, podemos ver al Internet como una vasta y conveniente fuente de procesamiento para la ejecución de aplicaciones distribuidas.

Esta forma de observar el Internet ya ha sido explotada en Computación Corporativa (17), computación en *Grid* (2) y redes *Peer–to–Peer* (15). Sin embargo, el uso del Internet como una fuente de metacomputación introduce nuevos problemas. Algunos de ellos provienen de la heterogeneidad de los sistemas participantes, de la dificultad de administrar ambientes de ejecución dinámicos, de los inconvenientes sobre la seguridad de los usuarios, y finalmente de los altos tiempos de retraso en la comunicación.

Con la finalidad de atenuar algunos de los problemas antes mencionados, nosotros proponemos una nueva infraestructura para el desarrollo de aplicaciones distribuidas en la Web. Esta infraestructura esta orientada ha permitir que todos los sistemas remotos conectados al Internet, puedan participar en la ejecución de dichas aplicaciones,

mediante una configuración bastante simple y con garantías de seguridad. Nuestra propuesta se basa exclusivamente en el uso de protocolos estándares para la *World Wide Web* y *applets* Java (20). Por ende, cualquier usuario que desee participar, requerirá únicamente de un navegador Web que tenga instalado y habilitado su JRE (*Java Runtime Environment*) correspondiente.

De esta manera, la heterogeneidad de los sistemas no es un problema, ya que el JRE es extremadamente portable. De hecho, este ambiente de ejecución es soportado por un conjunto innumerable de sistemas operativos y arquitecturas. De la misma forma, las dificultades para el manejo de un ambiente dinámico son drásticamente reducidas. No existe la necesidad de instalar programas adicionales, o de configurar a cada usuario de forma particular y/o manual. Finalmente, las preocupaciones de seguridad se eliminan, ya que el JRE toma control sobre todas las operaciones que un *applet* puede realizar y gracias a su *Java SandBox* (20) el código es ejecutado en un ambiente seguro de acuerdo al estándar que posee Java para ejecutar *applets*.

Un último problema por resolver, mencionado anteriormente, son los altos retardos en las comunicaciones por Internet, comparados con aquellos obtenidos en redes de área local. A pesar que esto se cambia constantemente debido a las nuevas infraestructuras existentes en las redes modernas, nuestra propuesta presenta un mecanismo que permite la comunicación directa entre componentes. Esta solución elimina las limitaciones impuesta por las políticas de Java dentro del JRE que impide a los *applets* conectarse hacia otra máquina que no sea el servidor de origen del cual partieron. Esta política restrictiva se la conoce como *"host–of–origin"*. Por ahora, esta política no puede ser modificada externamente, pero se puede convertir el código inseguro, para el cual se crearon este tipo de políticas, y transformalo en algo que pueda ser sujeto de confianza y comprobación. Para lograr esta característica, debemos firmar digitalmente nuestros *applets*, con lo cual los usuarios que revisando nuestro Certificado Digital deseen participar, nos permitirán acceder a privilegios no permitidos para *applets* no firmados y por ejemplo conectarnos a cualquier máquina que deseemos. Sin embargo esto conlleva

otros problemas, ya que sistemas que se encuentren en redes privadas (por ejemplo, detrás de un NAT o PAT), o que simplemente no confíen en el dueño del Certificado Digital, no podrían integrarse al sistema de procesamiento. Nuestra propuesta contempla estos casos y presenta una forma de comunicación, que cumpliendo con las normas del *Java Sandbox*, permite la transferencia de información entre clientes indirectamente.

La evaluación de nuestra propuesta se la realizó en base a tres aplicaciones típicas en sistemas distribuidos, además de dos *micro–benchmarks* orientados a medir características específicas de la comunicación. Los resultados de nuestra evaluación muestran que la escalabilidad para aplicaciones *coarse-grain* es muy atractiva. En los resultados de nuestros *micro–benchmarks* tenemos que la latencia se reduce en casi 30 % cuando utilizamos comunicación directa en lugar de comunicación indirecta entre componentes. Por otro lado, la velocidad de transferencia se mantiene constante en ambas alternativas de comunicación.

Capítulo 2

Fundamentos Teóricos

En este capítulo discutiremos algunos conceptos que dan soporte al tema principal de nuestra investigación. Específicamente nos gustaría revisar conceptos e ideas relacionados a la *World Wide Web*, la plataforma de Java y las firmas digitales.

2.1 La *World Wide Web*

La *World Wide Web*, o simplemente Web, es el servicio más usado en Internet. La Web se encuentra conformada por los *servidores Web*, los cuales almacenan y diseminan su contenido, conocido como *páginas Web*, las que son básicamente un conjunto de “tags” que definen como los datos serán expuestos en pantalla. Las páginas Web son accesibles a través de navegadores Web como: Internet Explorer, Netscape, Safari, Opera, Firefox, etc. Estos navegadores Web se comunican con los servidores por medio de un protocolo conocido como HTTP (*HyperText Transfer Protocol*). Este protocolo permite a los navegadores obtener las páginas Web almacenadas en cada servidor, además facilita el envío de información hacia los servidores. La versión más común de HTTP es la version 1.1, la cual se encuentra definida en el RFC-2616 (11). Las versiones más nuevas de los navegadores Web soportan en su totalidad el estándar HTTP/1.1.

Las páginas Web son localizadas por medio de un URL (*Uniform Resource Locator*), el cual no es más que una dirección identificada por medio de `http:` para acceso

mediante el protocolo HTTP. Pero no es el único identificador de protocolo soportado por los navegadores Web, estos usualmente también permiten conexiones del tipo `ftp:` para FTP (*FileTransfer Protocol*), `rtsp:` para RTSP (*Real-Time Streaming Protocol*), y `https:` for HTTPS (Una versión criptografiada de HTTP usando SSL).

Como se mencionó anteriormente, los servidores Web guardan páginas Web además de otros tipos de contenidos como son los gráficos, animaciones, *applets*, y otros elementos multimedia; permitiendo almacenar y distribuir diferentes contenidos mediante conexiones HTTP. Esto ha permitido el surgimiento de lo que se conoce como páginas Web "ricas", las cuales presentan de forma simultánea y creativa diferentes tipos de contenidos. Por su lado los navegadores dibujan las páginas en la pantalla llamando automáticamente a los programas necesarios para desplegar ese contenido adicional. Por ejemplo, en el caso de audio y video, los navegadores pueden requerir la instalación de algunos *plug-ins*, o también pueden recurrir a los propios sistemas de audio y video con los que cuenta el sistema operativo nativo.

2.2 La Plataforma Java

El Java SE (*Standard Edition*) es un ambiente completo para el desarrollo y ejecución de applicaciones desarrolladas en Java. Básicamente, son dos componentes principales los que lo forman, el JRE (*Java SE Runtime Envioroment*) y el JDK (*Java SE Development Kit*). El primero provee la máquina virtual de Java (sus siglas en inglés, JVM), el API (*Application Programming Interface*), además de una serie de componentes necesarios para poder ejecutar las aplicaciones Java. El segundo elemento permite el desarrollo de nuevas aplicaciones escritas en el lenguaje Java. Escencialmente es un conjunto de compiladores y herramientas de desarrollo.

En este trabajo nos enfocamos mayormente en el uso de *applets*, que son componentes de software escritos en el lenguaje Java y diseñados específicamente para ejecutar en el contexto de un navegador Web. Los *Applets* pueden ser fácilmente incluidos en

páginas Web y ejecutados por los navegadores que tengan el *plug-in* adecuado. En otras palabras los navegadores deben tener un JRE instalado y habilitado para la ejecución de *applets*.

Cuando un navegador es usado para ver páginas cuyo contenido es un *applet*, el código ejecutable del mismo es transferido desde el servidor Web hacia el navegador, luego es ejecutado localmente en la máquina virtual del mismo navegador, dentro del *Java Sandbox*. Este último componente se ocupa de restringir e imposibilitar que el código ejecutado realice acciones que el usuario no desee. El *Java SandBox* depende principalmente de tres elementos: el *Byte-code Verifier*, el *Class Loader*, y el *Security Manager*. Juntos, efectúan verificaciones de carga, ejecución y acceso a los diferentes elementos de la máquina local, al igual que al navegador.

El *Byte-code Verifier* revisa el código antes que este pueda ejecutarse. Este esquema de verificación esta diseñado para asegurar que el código se ejecute de acuerdo a reglas preestablecidas. El *Class Loader* determina cuando y como un *applet* puede agregar clases cuando se encuentra ejecutando. Parte de su trabajo es asegurarse que secciones del JRE no sean remplazadas por código ajeno que el *applet* esté intentando modificar. Finalmente el *Security Manager* restringe la forma en como el *applet* puede utilizar las interfaces accesibles. Este es un módulo que puede realizar pruebas en tiempo de ejecución sobre los métodos catalogados como peligrosos dentro del *applet*, y tiene la potestad de rechazar las llamadas y generar excepciones de seguridad.

En el caso de los *applets*, las políticas adoptadas por el *Security Manager* son de carácter restrictivo, permitiendo sólo un conjunto limitado de acciones, como es el caso de restringir conexiones a la máquina de origen. Para permitir que un *applet* pueda realizar acciones fuera de las permitidas en una máquina cliente, éste debe estar firmado digitalmente, y el cliente debe aceptar (confiar) el certificado digital por medio de un diálogo de confirmación. En caso de no ser aceptado, el *applet* no podrá ejecutarse.

2.3 Firmas Digitales

Una firma digital es un mecanismo criptográfico que garantiza la autenticidad, integridad y no–rechazo (*non-repudiation*) del mensaje. La autenticidad permite al receptor del mensaje confirmar la identidad del emisor. La integridad garantiza que el mensaje no ha sido alterado desde su creación hasta que llego a su destino. Finalmente, el no–rechazo (non-repudiation) impide que el emisor niegue la asociación de un mensaje con su persona.

En el sistema de criptografía de llave pública, cada usuario genera dos llaves: una pública y una privada. La llave púbica es distribuida libremente y no requiere de mecanismos adicionales de seguridad para su transmisión, por otro lado, la llave privada es conservada en secreto de forma segura. Una condición importante del sistema es la imposibilidad de obtener o generar la llave privada a partir de la llave pública. Basados en este sistema, un esquema general de firma digital consiste de tres algoritmos: uno para generar las llaves, uno que firma un documento en base a la llave privada, y uno para la verificación del contenido del documento por medio de la llave pública.

Por ejemplo, consideremos la siguiente situación ficticia en la cual Bob envía un mensaje a Alicia, y ella desea conocer si Bob realmente envio dicho mensaje. Bob al enviar el mensaje agrega su firma digital. La misma que es creada en base a la llave privada de Bob, y toma la forma de un simple valor numérico (normalmente representado como una cadena de caracteres). Al recibir el mensaje, Alicia puede comprobar que realmente proviene de Bob por medio de la llave pública de éste y el algoritmo de verificación. En caso de que el algoritmo indique que el mensaje es válido, Alicia puede tener la seguridad de que dicho mensaje proviene efectivamente de Bob, ya que el proceso de firmado esta diseñado para que su creación por externos, que no posean la llave privada correspondiente, sea imposible de producir.

La distribución de llaves públicas se la realiza por medio de certificados digitales (12), que para que sean realmente seguros deberán provenir de una entidad conocida. Esta entidad se denomina *Certificate Authority*, y es la que valida que la llave pública

provenga efectivamente de quien indica dicho certificado. Existen múltiples niveles de certificación, cada uno de los cuales requiere mayor o menor evidencia de que la información provista es la correcta y de que realmente pertenece a la persona que pide dicha certificación.

2.3.1 Firma Digital de *Applets*

Con el fin de poder realizar una comunicación directa entre los componentes de nuestras aplicaciones, es necesario firmar digitalmente los *applets* correspondientes. El proceso de firmado de un *applet* es bastante simple, pero es necesario primero obtener un certificado digital emitido por una entidad de confianza. Para nuestra evaluación no es necesario usar un certificado válido externamente, nos bastará con un certificado propio auto–firmado. Este certificado (auto–firmado) no puede ser verificado o confirmado por una entidad externa, por ende su uso no es adecuado para ambientes de producción. Sin embargo, es suficiente para un ambiente de desarrollo como el nuestro.

De esta forma, hemos usado el comando `keytool` para generar nuestro propio certificado auto–firmado. Para poder firmar nuestro código es necesario que éste se encuentre dentro de un archivo *jar*, el cual creamos usando el comando `jar`. Una vez realizado esto, procedemos a firmar el contenido con el comando `jarsigner`, el cual recibe como uno de sus parámetros el certificado a usar.

Capítulo 3

Infraestructura

Acorde a lo discutido anteriormente, lo que proponemos en este trabajo es una arquitectura genérica para el desarrollo y ejecución de aplicaciones distribuidas basadas en *applets*. Empezaremos describiendo las características que no fueron tomadas en cuenta para el desarrollo de este trabajo. Después, revisaremos los diversos elementos de nuestra infraestructura y cómo éstos interactúan entre si. Finalmente, describiremos brevemente, a manera de ejemplo, cada una de las aplicaciones usadas en nuestra evaluación.

3.1 Limitaciones

Es importante detallar de forma breve los elementos que no fueron considerados para el desarrollo de esta investigación, ya que sin estos se lo puede considerar inconcluso ó carente de funcionalidades. Sin embargo, son tópicos que fueron dejados de lado y que podrían ser implementados en versiones futuras de esta investigación.

- No se considera los casos en los cuales los clientes no son confiables en cuanto a su permanencia dentro del entorno de ejecución (escenario con clientes estables).
- No se consideran casos en los cuales las redes fallen, impidiendo una transmisión adecuada de la información (escenario con conectividad perfecta).

- No se cuenta con un servidor de nombres completamente funcional. La implementación actual es parcial, cubre los requisitos mínimos necesarios para el funcionamiento de nuestras aplicaciones.

3.2 Arquitectura

3.2.1 El *Broker*

Este componente es el centro de comunicación de nuestras aplicaciones distribuidas, ya que reenvía de forma transparente los mensajes entre los diferentes módulos. Sin embargo, existen algunas funcionalidades que se le pueden agregar, como por ejemplo un sistemas de identificación (*name server*), soporte para memoria compartida y manejo automático de los errores de comunicación.

La implementación actual de nuestro *broker* es bastante genérica, sirve para varias aplicaciones, y se encuentra implementado en Java. Sin embargo, podría haber sido implementado en otro lenguaje de programación con capacidades de red. La única característica requerida es que debe ejecutar en la misma máquina donde opera el servidor Web. Adicionalmente, se puede mencionar como una característica especial, que nuestro *broker* puede ser usado por más de una aplicación distribuida simultáneamente.

El *broker* actual, una vez que a sido levantado y se encuentra ejecutando, tiene dos funciones básicas: entregar IDs únicos a cada cliente que se conecta, y enrutar de forma transparente las comunicaciones de los diferentes módulos basado en los identificadores entregados. Estas funciones deben ser ejecutadas simultáneamente, ya que múltiples módulos pueden querer comunicarse al mismo tiempo o ingresar al sistema cuando otros (módulos) ya se encuentran operando. Para conseguir esto, se maneja un sistema *multithreaded* donde se mantiene un *thread* por cada cliente. De esta forma, se pueden cubrir las demandas de varios clientes simultáneos.

Por estas y otras características de implementación, el sistema actualmente sólo soporta mensajes asincrónicos y no posee mecanismos sofisticados para el manejo de

errores. Sin embargo, la implementación es lo suficientemente simple y robusta como para ser expandida e implementada en cualquier lenguaje de programación.

3.2.2 El Coordinador

Este es el elemento que tiene a su cargo el control de una aplicación distribuida específica. Cuando es iniciado, este comienza conectándose al *broker*, y una vez conectado, espera por los clientes a que se reporten. Al llegar los clientes, este elemento les envía los parámetros computacionales de cada uno. Cuando un cliente termina sus cálculos, envía los resultados hacia el coordinador, el cual dependiendo de la aplicación, almacena y procesa los resultados de manera diferente.

Una característica interesante del coordinador es que no requiere ser ejecutado en la misma máquina usada por el *broker* y el servidor Web. Así, el coordinador puede ser simplemente tratado como otro cliente remoto del sistema.

Dentro de este trabajo se implementaron tres coordinadores diferentes, cada uno correspondiente a una aplicación distribuida diferente. Estos fueron creados para cubrir los aspectos básicos de funcionamiento de cada aplicación y con la finalidad de probar nuestros conceptos; Los mismos serán detallados más adelante cuando se describa cada aplicación.

Los coordinadores desarrollados para este proyecto fueron codificados usando Java, pero este no es un requisito, ya que puede ser codificado en cualquier lenguaje de programación con capacidades de comunicación vía red. Se recomienda también que cada coordinador implemente algunos mecanismos de tolerancia a fallas, ya que los clientes no son confiables en el sentido de permanecer funcionales por un tiempo específico. Estos mecanismos de tolerancia a fallas no son parte de este trabajo, como fue explicado en 3.1.

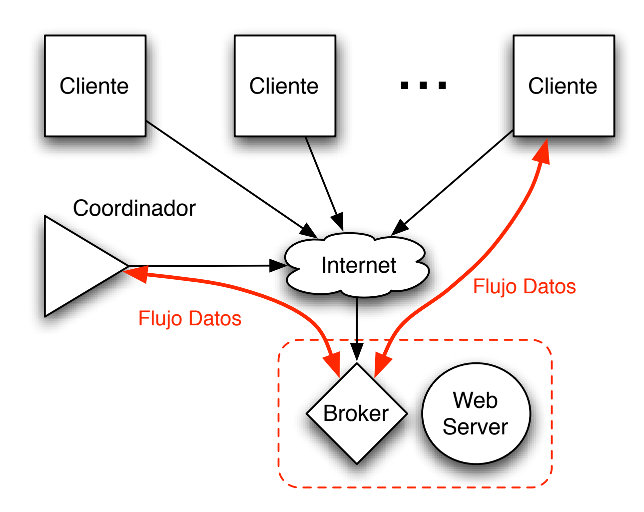


Figura 3.1: Estructura de una aplicación distribuida típica.

3.2.3 Los Clientes

Los clientes son las entidades que realmente realizan el procesamiento de los datos dentro de una aplicación distribuida. Cada cliente es ejecutado como un *applet* incluido en una página Web. Es así como los mismos pueden ser fácilmente almacenados y distribuidos a través de un servidor Web. Una máquina dispuesta a participar de la ejecución de una aplicación distribuida necesita únicamente ingresar a la página Web apropiada, con un navegador que este habilitado para ejecutar *applets*.

Una visión general de la estructura de una aplicación distribuida que use nuestra infraestructura es presentada en la figura A.1. En la misma podemos ver como los clientes se conectan entre si y con el coordinador a través del *broker*.

La mayor restricción de los clientes se encuentra en la imposibilidad de comunicarse con módulos que no esten ubicados en el servidor del cual fue descargado el *applet*. Por ende, si deseamos mantener el esquema de seguridad es necesario tener un *broker* en el servidor Web para tranferencia de mensajes. De esta manera, para poder realizar cualquier tarea de comunicación el cliente deberá conectarse al *broker* y enviar sus mensajes con el formato apropiado descrito en la subsección 3.2.6. Estas limitaciones pueden ser eliminadas por el correspondiente certificado digital que avale la seguridad de dicho código.

3.2.4 El Servidor Web

Este servidor es el único elemento del sistema que no fue desarrollado completamente por nosotros, pues no era uno de los objetivos del trabajo. Lo que se hizó fue tomar un servidor Web de código libre e instalarlo. El servidor seleccionado fue `mathopd`, en su versión `1.5p5`. Sin embargo, es necesario aclarar que se puede utilizar cualquier otro servidor Web estándar para la plataforma de desarrollo.

La única funcionalidad requerida en este elemento es la de almacenar las páginas Web de la aplicación y sus elementos de ejecución correspondientes. Estos elemento serán luego accedidos por los clientes Web de nuestra infraestructura.

3.2.5 Biblioteca de Desarrollo

Con el fin de proveer una infraestructura genérica de fácil utilización, es importante soportar la creación de nuevas aplicaciones mediante bibliotecas de funciones. Debido a esto, nuestra infraestructura cuenta con una biblioteca Java de fácil utilización para el desarrollo de nuevas aplicaciones. La idea principal de la misma es facilitar el desarrollo de nuevos elementos usando como base los componentes aquí descritos, que por cierto, ya han sido probados también.

De esta manera, cualquier programador interesado en el desarrollo de nuevas aplicaciones distribuidas pueden extender lo presentado aquí de acuerdo a sus necesidades sin necesriamente partir de cero.

3.2.6 El Formato de los Mensajes

Cada aplicación tiene un modo diferente de interactuar entre sus módulos, obviamente esto tiene que ver con los requisitos propios de cada aplicación. Pero el formato básico de los mensajes para comunicación es el mismo en todos los casos, ya que la idea principal de este formato es simplemente permitir el traspaso de mensajes entre módulos de forma transparente. Pero antes de describir el formato de los mensajes, deberemos

primero definir el formato de los identificadores de módulo. Cada módulo conectado al *broker* tiene un identificador único, el cual es provisto por el *broker* al momento de la conexión. Este identificador se encuentra definido como un byte, dando un total de hasta 256 módulos que pueden estar conectados al *broker*. Sin embargo, cambiar este byte por un entero, y permitir un mayor número de clientes, es bastante simple.

Así, el formato de los mensajes consiste de un byte de identificación para el destino y un entero que indican la longitud del mensaje, para luego continuar con el número de bytes del mensaje indicados en el entero. Una vez que el mensaje es recibido por el *broker*, este último sobre–escribe el primer byte del mensaje y coloca el identificador del origen en lugar del identificador del destino, con lo que el receptor del mensaje puede deducir quien es el que le envía el mensaje para dar la respuesta apropiada.

Para los casos en los cuales no se utiliza el *broker* en el paso de mensajes, el proceso anteriormente descrito cambia un poco. Debido a la forma como se conectan los módulos (a través de conexiones TCP/IP directas, ver subseccón 3.2.8), cada uno conoce con quien se encuentra conectando sin la necesidad de requerir los elementos de identificación antes mencionados. Lo que si se mantiene es el envío de la longitud del mensaje representada por medio de un entero.

3.2.7 Comunicación Indirecta

De lo descrito anteriormente tenemos que el método principal de comunicación entre los componentes de una aplicación es a través del *broker* que actúa como un elemento intemediario en la transmisión de los mensajes. En este esquema se cumplen estrictamente las reglas dispuestas por Java para la ejecución de *applets* en clientes remotos. Esto significa que no podemos comunicarnos con otro servidor que no sea el de origen, y por ende tenemos al *broker* para que re–dirija los mensajes entre componentes.

El esquema descrito tiene la ventaja principal de permitir la participación de cualquier máquina conectada al Internet, sin importar si se encuentran detrás de un *proxy*, *firewall*, NAT o PAT. Esto se debe a que el tráfico generado por los clientes se ve como

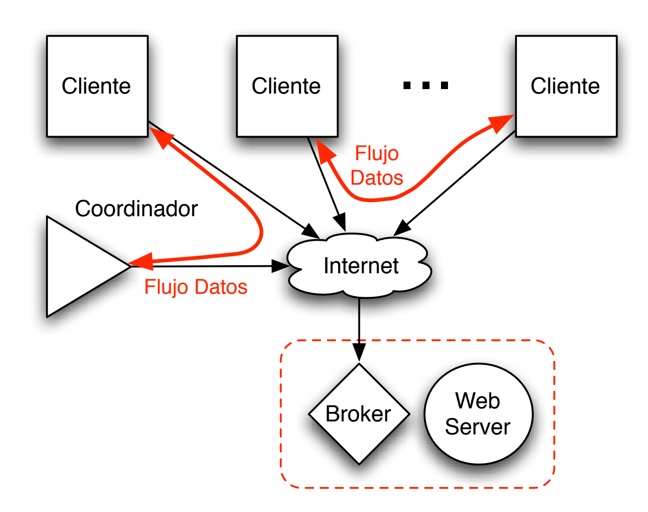


Figura 3.2: Interacción de los componentes usando comunicación directa.

cualquier otro tráfico Web hacia Internet desde la máquina de un usuario específico y no requiere de ningún acceso directo especial.

Su principal desventaja es el hecho de pasar todo el contenido de los mensajes por el *broker*, lo que puede acarrear retrasos en la transmisión debido a múltiples circunstancias. Como ejemplos podemos mencionar la sobrecarga de envíos y recepciones en el *broker*, y la duplicación de la latencia normal de comunicación. Inclusive, una falla en el *broker* podría hacer fracasar todo una aplicación por ser el elemento central de nuestra infraestructura.

3.2.8 Comunicación Directa

Tomando en cuenta los problemas de latencia que puede causar la comunicación indirecta y el hecho de que algunas aplicaciones pueden sobrecargar la capacidad del *broker*, se implementó un método por el cual se permite que los clientes se comuniquen entre ellos directamente sin la necesidad del *broker*. Con la finalidad de mantener la seguridad, se decidió usar el sistema de certificados digitales, el cual nos permite sobrepasar las barreras impuestas por Java.

La dificultad de este esquema es que sólo funciona con clientes que posean acceso directo a Internet. En otras palabras, no funciona con máquinas detrás de un NAT o

PAT (básicamente, que no poseen una dirección IP ruteable).

El diagrama presentado en la figura 3.2, muestra como los componentes de una aplicación interactúan usando comunicación directa.

3.3 Ejemplos de Aplicaciones

Ahora que conocemos los elementos con los cuales se construyen nuestras aplicaciones distribuidas, podemos pasar a describir cada aplicación y cómo operan. Tomando en cuenta que casi todos los elementos son bastante genéricos, nos restringiremos a describir fundamentalmente cómo funcionan los coordinadores y sus clientes, ya que estos son las partes más específicas de cada aplicación.

3.3.1 Integración Numérica

En esta implementación particular de integración numérica, el coordinador de la aplicación empieza dividiendo el intervalo a ser integrado en N partes, para luego esperar a que lleguen los pedidos de los clientes. Cuando uno de ellos pide los parámetros de la integración, este recibe los límites superior e inferior del intervalo correspondiente, así como el incremento Δ con el cual debe integrarse. Estos parámetros son usados para integrar la función definida en los clientes a través del método de Newton–Coates. Una vez terminada la integración en el cliente, el resultado es devuelto al coordinador. El mismo que, por su parte, recibe todos los resultados parciales y una vez que todos los clientes han terminado, suma esos resultados para dar la respuesta final de la aplicación.

Como podemos observar, esta es una aplicación extremadamente paralelizable, ya que la comunicación necesaria con el coordinador es mínima y cada cliente es completamente independiente de los demás, no requiriendo comunicación entre los mismos. De hecho, cada intervalo asignado a un cliente es integrable en cualquier orden, sin que se altere el resultado final de la aplicación.

3.3.2 Multiplicación de Matrices

Para el caso de la multiplicación de matrices, el coordinador empieza por establecer las matrices A y B a ser multiplicadas. Luego, este divide la tarea en N partes, de las cuales a cada cliente le toca un conjunto de filas de A y columnas de B. Terminado esto, el coordinador espera por los pedidos de los clientes. Cuando uno de ellos solicita los parámetros de la aplicación, el coordinador le envía el identificador del trabajo a realizar (básicamente, las coordenadas del resultado), así como las filas y columnas asociadas a dicho identificador. El cliente entonces calcula

$$r_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}$$

con los datos recibidos. Cuando termina, el cliente envía de vuelta el identificador de la tarea y los resultados correspondientes. Por su parte el coordinador recibe los resultados y los almacena (en base a los identificadores) en el matriz de salida. Una vez recibido todos los resultados desde los clientes, el coordinador imprime la matriz resultado y termina su ejecución.

Como en el caso de la integración numérica, esta aplicación es fácilmente paralelizable. Esto se debe principalmente a que no se requiere comunicación entre los clientes, pues estos no dependen los unos de los otros para terminar su tarea asignada. Sin embargo, existe mayor comunicación entre el coordinador y los clientes debido a que se envía parte de las matrices A y B desde el coordinador hacia los clientes, y estos últimos retornan una serie de resultados parciales. Estas transferencias de datos son de mayor tamaño y pueden convertirse en un problema dentro del sistema.

3.3.3 Sobre–Relajación Sucesiva

En nuestra implementación de SOR (*Sucessive Over–Relaxation*), el coordinador comienza definiendo la matriz de elementos a procesar, la cual se divide en N partes contiguas. A cada parte se le agregan las filas superior e inferior adyacentes, según sea

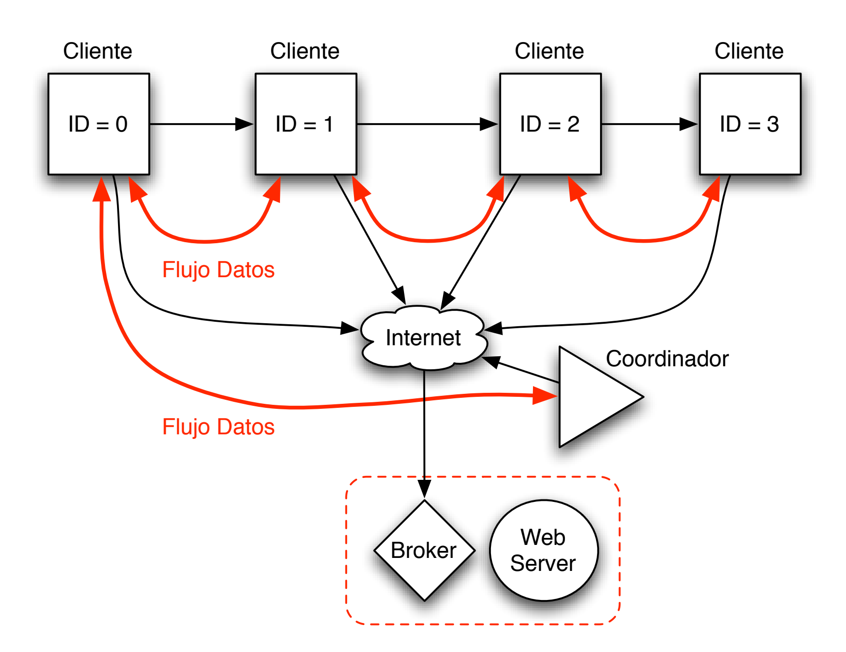


Figura 3.3: Configuración de SOR cuando se utiliza comunicación directa.

el caso. Una vez realizado esto, se procede a esperar los pedidos. Cuando los clientes llegan a pedir una tarea, el coordinador les asigna una parte de la matriz. Al terminar la recepción, cada uno empieza a ejecutar la siguiente operación reiterativamente:

$$a_{i,j} = (a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1})/4.$$

Luego de terminada cada iteración, los mismos deben intercambiar con sus vecinos las filas superior e inferior con la finalidad de actualizar las condiciones de borde y poder continuar con la siguiente iteracción. Terminadas todas las iteracciones, cada cliente retorna sus resultados parciales al coordinador, el mismo que los almacena en la matriz de salida. Una vez que todos los clientes han enviado sus resultados correspondientes, el coordinador imprime la matriz resultante y termina su ejecución.

Como se puede ver, esta aplicación tiene un alto grado de comunicación entre los clientes, y de hecho no es fácilmente paralelizable. Como fue mencionado, luego de cada iteracción es necesario intercambiar las filas superior e inferior con los clientes adyacentes, esto incluye tanto el envío como recepción de 2 filas de la matriz.

Con esta funcionalidad en mente, se decidió implementar dos versiones de esta

aplicación, la primera utilizado comunicación indirecta a través del *broker*, y la segunda con comunicación directa para reducir la latencia de comunicación entre los clientes. Por el método de comunicación directa, la transmisión de las filas luego de cada iteracción se la realiza directamente hacia el cliente que las requiere, con el fin de eliminar la necesidad del *broker* para su intercambio. Como se requiere conocer la ubicación de los clientes adyacentes para establecer las conexiones directas, se ha creado una etapa de inicialización adicional dentro del coordinador para transferir estas informaciones a los clientes correspondientes.

Capítulo 4

Evaluación

En este capítulo presentaremos los resultados de desempeño de nuestros *micro–benchmarks* y de las tres aplicaciones discutidas en la sección 3. Pero antes, describiremos brevemente la metodología usada para realizar estas pruebas y como funcionan nuestros *micro–benchmarks*.

4.1 Metodología

Todos los experimentos de desempeño fueron ejecutados en una red local con cuatro estaciones de trabajo Sun Blade 1500 y dos servidores Sun Fire V240. Todas las máquinas (incluidos los servidores) se encuentran ejecutando el sistema operativo Solaris 9/04 y están interconectadas mediante una red Fast-Ethernet formada por un *switch* Cisco Catalyst 2950. Sobre esta plataforma fueron ejecutados los dos *micro–benchmarks* y las tres aplicaciones paralelas.

4.2 *Micro–benchmarks*

Con el propósito de evaluar la latencia y velocidad de transferencia entre módulos en la infraestructura propuesta, hemos ejecutado dos evaluaciones básicas (*micro–benchmarks*). La estructura usada en estas evaluaciones es prácticamente la misma

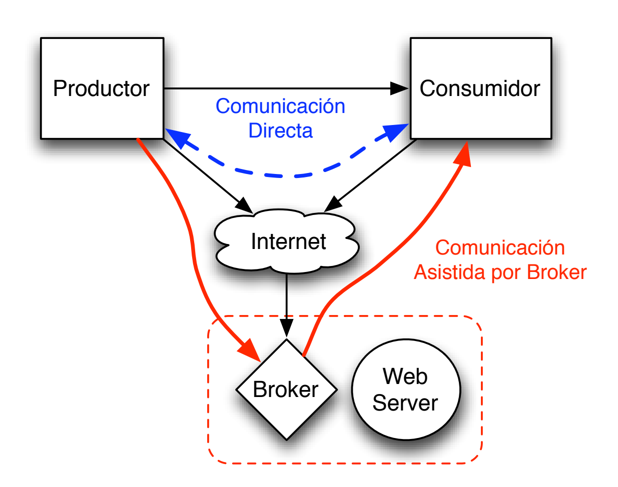


Figura 4.1: Estructura usada por los *micro–benchmarks*.

que la descrita en la sección 3, con la diferencia de que no necesitamos un elemento coordinador debido a la simplicidad de los *micro–benchmarks*. En la figura 4.1 podemos observar la estructura usada en nuestras evaluaciones, donde un productor y un consumidor se conectan al *broker* y no existe un coordinador. Los módulos productor y consumidor pueden conectarse a través del *broker* o pueden usar comunicación directa entre ellos.

La evaluación de latencia consiste en el envío de un mensaje de 0–bytes desde el productor hacia el consumidor para luego esperar por la correspondiente respuesta. Este proceso se repite 10 mil veces y el tiempo promedio de cada envío es medido. Por otro lado, para medir la velocidad de transferencia, el productor envía hacia el consumidor mensajes de 1024–bytes sin esperar por respuesta o confirmación alguna. Luego de enviar 10 mil veces dicho mensaje, la velocidad promedio es calculada.

4.3 Resultados

Como fue mencionado, se ejecutaron dos *micro–benchmarks* y tres aplicaciones reales usando comunicación por medio del *broker* y en forma directa entre los *applets*.

Comunicación	Latencia (ms)	Tasa de Trans. (KB/s)
Usando el *broker*	0.30	11.0
Directa	0.22	11.6
Mejora	27 %	5 %

Cuadro 4.1: Resultados de los *micro–benchmarks.*

4.3.1 Resultados de los *Micro–benchmarks*

En la tabla 4.1 se encuentran resumidas la latencia y velocidad de transmisión para nuestros *micro–benchmarks*. Como podemos observar, la latencia en la comunicación directa es 27 % menor que la comunicación por medio del *broker*. Con relación a la velocidad de transferencia, la diferencia entre los dos esquemas de comunicación es poco significativa. En ambos casos, las interfases Fast–Ethernet (100–Mbps $\sim$ 12.5MB/s) prácticamente se saturan.

De esta manera, podemos concluir que la comunicación directa entre *applets* es de gran utilidad en términos de latencia. Sin embargo, la velocidad de transmisión entre módulos puede ser fácilmente mejorada usando comunicación directa cuando el *broker* se satura y se convierte en un verdadero cuello de botella. Para ilustrar esta situación, consideremos el caso donde múltiples *micro–benchmarks* se encuentran ejecutando simultáneamente sobre el mismo *broker*. En este escenario, cada evaluación está compartiendo el ancho de banda total provisto por el *broker* y los resultados medidos por cada una de las evaluaciones serán inferiores a los obtenidos en la tabla 4.1.

4.3.2 Resultados de Aplicaciones Reales

La figura 4.2 muestra el incremento de desempeño para el caso de la integración numérica usando hasta 4 estaciones de trabajo actuando como clientes. Se puede observar que el incremento de desempeño es perfectamente lineal, debido a que no existe mucha comunicación entre los componentes de la aplicación.

De forma similar, la figura 4.3 muestra el incremento de desempeño para la multi-

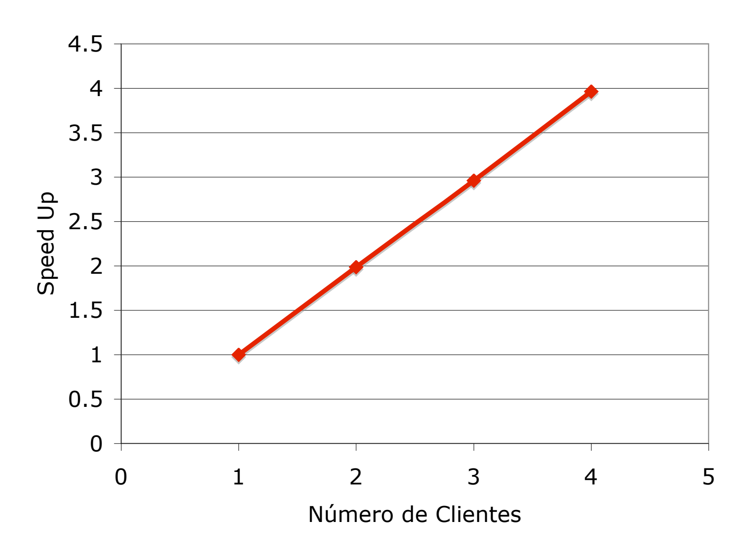


Figura 4.2: Incremento de desempeño para la integración numérica.

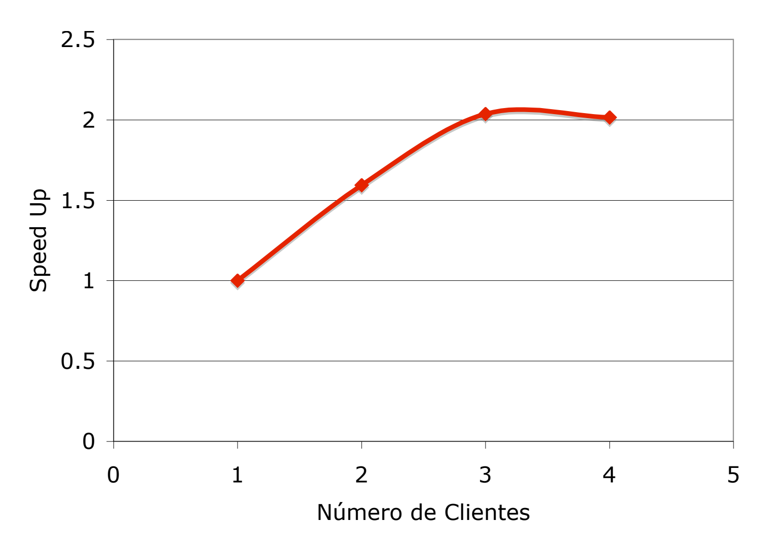


Figura 4.3: Incremento de desempeño para la multiplicación de matrices.

plicación de matrices. La escalabilidad de esta aplicación no es tan buena como el caso de la integración numérica, ya que la comunicación con el coordinador se convierte en un cuello de botella. Recordando la descripción de las aplicaciones hecha en el capítulo 3, el coordinador necesita enviar dos regiones bastantes grandes de las matriz a cada cliente y esperar por los resultados parciales (otra matriz). Estas dos operaciones, la distribución de los datos y la recopilación de resultados, serializan la ejecución de la aplicación en una fracción significativa de su tiempo de ejecución.

El incremento de desempeño para la versión de SOR que usa comunicación a través del *broker* es presentado en la figura 4.4. Esta aplicación fue ejecutada usando hasta



Figura 4.4: Incremento de desempeño para SOR usando comunicación indirecta.

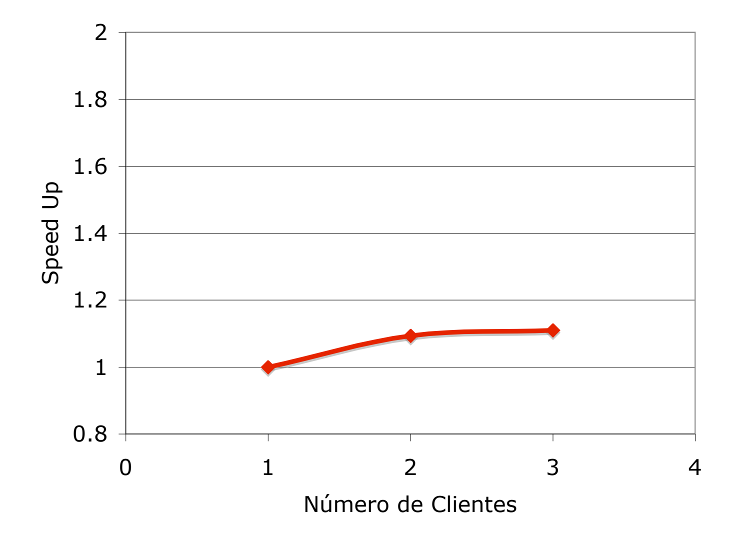


Figura 4.5: Incremento de desempeño para SOR usando comunicación directa.

cuatro estaciones de trabajo que actúan como clientes. En la figura se puede observar que prácticamente no existe incremento de desempeño. De hecho, usando cuatro nodos de computación sólo reducimos el tiempo de ejecución en aproximadamente 15 %. Este comportamiento puede ser explicado por la fuerte sincronización necesaria en esta aplicación, además del costo extra requerido por la distribución y recopilación de datos.

El incremento de desempeño para la segunda versión de SOR, en este caso usando comunicación directa entre *applets*, es mostrado en la figura 4.5. Tenemos que para este caso, el desempeño mejora notoriamente con dos clientes, pero en general es bastante bajo y poco significativo. La explicación para este comportamiento se encuentra una

vez más en la fuerte sincronización requerida por la aplicación, además del costo extra introducido por la transmisión de datos y la necesaria recopilación de resultados.

Capítulo 5

Trabajos Relacionados

Existen múltiples trabajos relacionados con la integración de computadores en un único recurso computacional global. Algunos de estos trabajos proporcionan funcionalidad de bajo nivel, como es el caso de PVM (13), MPI (9), TreadMarks (1) y Brazos (19). En el caso de PVM y MPI, estos son sistemas de paso de mensajes portables, mientras que TreadMarks y Brazos proveen un ambiente distribuido con un único espacio de memoria usando máquinas débilmente acopladas. Estos sistemas requieren el mantenimiento de los archivos binarios para cada arquitectura usada en la computación y el programa paralelo debe residir en cada máquina que desee ingresar al sistema (o en un sistema de archivos compartido). Adicionalmente, se requiere contar con un usuario funcional en cada uno de los sistemas participantes. Todos estos factores dificultan y limitan su uso como un sistema de meta–computación para la Web. Inclusive, no existe soporte para balanceamiento de carga ni tolerancia a fallas en los mencionados sistemas.

Entre los trabajos basados en la plataforma Java para crear sistemas distribuidos tenemos JPVM (10), JMPI (8), ATLAS (3), ParaWeb (7), JavaParty (18), SMPD Programming in Java (14), WebFlow (6), Ninflet (21), Charlotte (5), Javelin (16) y KnittingFactory (4).

JPVM y JMPI usan Java para resolver la heterogeneidad de los sistemas, pero no están diseñados para ejecutar en máquinas no configuradas previamente. Estos sistemas

proveen un esquema para el paso de mensajes entre aplicaciones Java que operan directamente sobre el sistema operativo (aplicaciones *stand–alone*). Por otro lado, ATLAS crea un entorno global de computación basado en Java, asegurando la escalabilidad de las aplicaciones a través de un sistema jerárquico de administradores. Se fundamenta en el uso de código nativo que elimina la seguridades de ejecución en el sistema. De igual forma, ParaWeb crea un entorno global de computación usando una extensión del ambiente de desarrollo Java por medio de bibliotecas de clases paralelas y del sistema de ejecución Java. Por ende, los usuarios necesitan instalar dichas extensiones para permitir que unidades de computación Java sean ejecutadas transparentemente en forma remota. Continuando con la descripción de estos sistemas, JavaParty contiene mecanismos (construidos sobre RMI) que permiten la distribución transparente de objetos remotos. Requiere que se ejecute un proceso local llamado *LocalJP* en todas la máquinas utilizadas. De forma similar, el uso del modelo de programación SMPD (*Single Program Multiple Data*) en un sistema distribuido con memoria compartida (14) requiere la ejecución de un proceso local Java que actúa como su ambiente de ejecución. Por otro lado, WebFlow es uno de los primeros desarrollos de un sistema de "workflow" que soporta un sistema centralizado en ambientes *Grid*. Este incluye un paradigma de programación y un modelo de coordinación completo para Java que excluye a los típicos usuarios de navegadores Web. Finalmente, Ninflet es una infraestructura para la migración de objetos que se enfoca en los sistemas de computación paralelos con ciclos libres. Los proveedores de recursos tienen la necesidad de ejecutar el *Ninflet Server Deamon* en todas sus máquinas.

Como podemos observar, la mayoría de los sistemas anteriores no permite que todos los sistemas de computación en Internet puedan participar de la ejecución de aplicaciones distribuidas usando únicamente navegadores Web convencionales. Todos los sistemas referenciados hasta el momento necesitan una cuenta local o por lo menos la posibilidad de ejecutar un proceso externo en cada máquina.

De los sistemas citados, los que permiten el uso de navegadores Web para la ejecu-

ción de aplicaciones distribuidas o paralelas son Charlotte, Javelin, y KnittingFactory. Estos sistemas fueron específicamente diseñados para la implementación de sistemas de computación paralela sobre la Web. Usando la habilidad de los navegadores Web para bajar y ejecutar *applets* remotos, estos sistemas proveen el medio por el cual cualquier usuario, en cualquier parte del Internet, usando cualquier plataforma en la que exista un navegador Web con capacidad Java, pueda ser parte del sistema de computación paralela. Estos tres sistemas presentan una serie de características similares de diseño; entre las principales podemos citar: requieren la ejecución de un servidor Web junto con una aplicación local Java en el nodo central del sistema. El rol de la aplicación Java en el nodo central es distribuir y coordinar el trabajo entre los navegadores Web, y actuar como un agente de traspaso de mensajes para la comunicación entre los navegadores.

KnittingFactory además de funcionar de la forma anteriormente descrita, puede también actuar de una forma completamente particular. Primero, tiene la posibilidad de violar las restricciones del *Java Sandbox* usando un RMI (*Remote Method Invocation*) propio[1]. Esta característica permite la conexión directa entre los *applets* que intervienen en la aplicación. Sin embargo, el sistema como tal no posee un servidor de nombres para localizar los diversos elementos dentro del ambiente. En segundo lugar, este sistema tiene la posibilidad de ejecutar los procesos de coordinacion, llamados *Initiators*, en cualquier máquina diferente al servidor original del cual partieron los *applets*.

Nuestra propuesta es similar a las de Charlotte y de Javelin, pero nosotros también permitimos la comunicación directa entre *applets* usando certificados digitales. De esta forma tenemos las ventajas de KnittingFactory, pero de forma mucho más simple y sobre todo, con mayor seguridad. Adicionalmente, nuestro trabajo incluye una evaluación detallada del desempeño de aplicaciones distribuidas reales y el uso de comunicación directa e indirecta entre *applets*.

[1]Este sistema sólo funciona en Java 1.1.

Capítulo 6

Conclusiones y Recomendaciones

6.1 Conclusiones

Este trabajo de desarrollo e investigación describe una infraestrutura genérica y al mismo tiempo simple para el desarrollo de aplicaciones distribuidas y paralelas en la Web. Los usuarios de esta infraestructura únicamente necesitan un navegador Web que soporte *applets* Java para la ejecución de las mencionadas aplicaciones. Adicionalmente, hemos desarrollado una biblioteca de clases en Java y un conjunto de aplicaciones de ejemplo que soportarán el desarrollo de nuevas aplicaciones. Básicamente, los desarrolladores necesitan escribir un coordinador para la aplicación y los *applets* clientes correspondientes. El *broker* de comunicación desarrollado en este trabajo puede ser reusado sin ninguna modificación. Finalmente, los *applets* clientes deben ser puestos a disposición del público a través del servidor Web ejecutando en la misma máquina donde el *broker* está corriendo.

Una propuesta fundamental en nuestro trabajo es el uso de comunicación directa entre *applets* usando firmas digitales. A lo largo de este trabajo se ha comprobado que es importante usar comunicación directa entre componentes cuando sea posible, ya que la latencia y velocidad de transferencia son significativamente influenciadas por el modelo de comunicación. Además, por ser un elemento centralizador en nuestra

infraestrutura, el *broker* puede convertirse fácilmente en un cuello de botella para la comunicación, o peor todavía, un punto de alta vulnerabilidad en la operación normal de las aplicaciones.

Por otro lado, reducir los requerimientos de comunicación entre los componentes de una misma aplicación siempre es útil para incrementar el desempeño de dichas aplicaciones. Las mejoras de desempeño obtenidas por las aplicaiones dependen de sus limitaciones en términos de computación y principalmente comunicación. Aquellas que presentan una granularidad bastante gruesa (*coarse–grain*) normalmente obtienen incrementos de desempeño lineales.

Finalmente, otros factores a tomar en cuenta son el balanceamiento de carga y la tolerancia a fallas, elementos clave para una operación adecuada y con buen desempeño de nuestras aplicaciones. Recordemos que la Web no es una plataforma de computación homogénea, peor aún confiable.

6.2 Recomendaciones

Tomando en cuenta los resultados obtenidos durante el desarrollo de esta infaestructura y las conclusiones previamente mencionadas, nos permitimos detallar algunas recomendaciones y trabajos futuros.

La biblioteca de funciones debería ser extendida en base a las nuevas experiencias que se tengan con el desarrollo de otras aplicaciones distribuidas y paralelas. El contenido de la biblioteca actual es un tanto limitado debido al conjunto básico de aplicaciones que se ha desarrollado en este trabajo.

El servicio de identificación y localización de módulos no corresponde al caso más general. Sería adecuado contar con un verdadero servidor de nombres (*name server*) incluido en el *broker* que facilite todas las tareas de identificación, asociación, nombramiento, descubrimiento de rutas y localización de los componentes pertenecientes a una aplicación.

Con relación al balanceamiento de carga y tolerancia a fallas, sería adecuado desarrollar técnicas de replicación que permitan ejecutar una misma tarea en dos o más clientes para garantizar su terminación a tiempo y la calidad de los datos generados por esa tarea específica. Igualmente, podrían mantenerse indicadores de la velocidad relativa de cada máquina y usar esos indicadores en las asignaciones futuras de tareas.

Finalmente, considerando las limitaciones y problemas previamente atribuidos al *broker*, sería conveniente investigar la posibilidad de usar múltiples *brokers* dentro de una misma aplicación. De esa forma, se incrementaría la confiabilidad y escalabilidad de las aplicaciones, especialmente cuando la comunicación directa entre *applets* no pueda usarse.

Bibliografía

[1] Christiana Amza, Alan L. Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, and Willy Zwaenepoel, *Treadmarks: Shared memory computing on networks of workstations*, IEEE Computer **29** (1996), no. 2, 18–28. 26

[2] Mark Baker, Rajkumar Buyya, and Domenico Laforenza, *Grids and grid technologies for wide-area distributed computing*, Software – Practice & Experience **32** (2002), no. 15, 1437–1466. 1

[3] J. Eric Baldeschwieler, Robert D. Blumofe, and Eric A. Brewer, *ATLAS: An infrastructure for global computing*, Proceedings of the 7th ACM SIGOPS European Workshop on System Support for Worldwide Applications, 1996. 26

[4] Arash Baratloo, Mehmet Karaul, Holger Karl, and Zvi M. Kedem, *An infraestructure for network computing with Java applets*, Concurrency: Practice and Experience **10** (1998), no. 11–13, 1029–1041. 26

[5] Arash Baratloo, Mehmet Karaul, Zvy M. Kedem, and Peter Wyckoff, *Charlotte: Metacomputing on the web*, Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems, 1996. 26

[6] Dimple Bhatia, Vanco Burzevski, Maja Camuseva, Geoffrey Fox, Wojtek Furmanski, and Girish Premchandran, *Webflow – a visual programming paradigm for Web/Java based coarse grain distributed computing*, Concurrency – Practice and Experience **9** (1997), no. 6, 555–577. 26

[7] Tim Brecht, Harjinder Sandhu, Meijuan Shan, and Jimmy Talbot, *ParaWeb: Towards World-Wide Supercomputing*, Proceedings of the 7th ACM SIGOPS European Workshop on System Support for Worldwide Applications, 1996. 26

[8] Kivanc Dincer, *Ubiquitous message passing interface implementation in Java: JMPI*, Proceedings of the 13th International Parallel Processing Symposium, 1998. 26

[9] Jack J. Dongarra, Steve W. Otto, Marc Snir, and David Walker, *A message passing standard for MPP and workstations*, Communications of the ACM **39** (1996), no. 7, 84–90. 26

[10] Adam Ferrari, *JPVM: Network Parallel Computing in Java*, Concurrency – Practice and Experience **10** (1998), no. 11-13, 985–992. 26

[11] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk, Larry Masinter, Paul J. Leach, and Tim Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, Tech. Report RFC 2616, Internet Engineering Task Force, June 1999. 4

[12] Morrie Gasser, Andy Goldstein, Charlie Kaufman, and Butler Lampson, *The digital distributed system security architecture*, Proceedings of the 12th NIST-NCSC National Computer Security Conference, 1989, pp. 305–319. 7

[13] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam, *PVM parallel virtual machine, A user's guide and tutorial for networked parallel computing*, MIT Press, Cambridge, Mass., 1994. 26

[14] Susan Flynn Hummel, Ton Ngo, and Harini Srinivasan, *SPMD programming in Java*, Concurrency: Practice and Experience **9** (1997), no. 6, 621–631. 26, 27

[15] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu, *Peer-to-Peer Computing*, Tech. Report HPL-2002-57R1, HP Labs, Palo Alto, CA, July 2002. 1

[16] Michael O. Neary, Bernd O. Christiansen, Peter Cappello, and Klaus E. Schauser, *Javelin: Parallel computing on the Internet*, Future Generation Computer Systems **15** (1999), no. 5–6, 659–674. 26

[17] Michael P. Papazoglou and Willem-Jan van den Heuvel, *Web services management: A survey*, IEEE Internet Computing **9** (2005), no. 6, 58–64. 1

[18] Michael Philippsen and Matthias Zenger, *JavaParty — transparent remote objects in Java*, Concurrency: Practice and Experience **9** (1997), no. 11, 1225–1242. 26

[19] Evan Speight and John K. Bennett, *Brazos: A third generation DSM system*, Proceedings of the USENIX Windows NT Workshop, 1997. 26

[20] Sun Microsystems Inc., *Java Technology*, `http://java.sun.com`, June 2006. 2

[21] Hiromitsu Takagi, Satoshi Matsuoka, Hidemoto Nakada, Satoshi Sekiguchi, Mitsuhisa Satoh, and Umpei Nagashima, *Ninflet: A migratable parallel objects framework using Java*, Concurrency: Practice and Experience **10** (1998), no. 11–13, 1063–1078. 26

Apéndice A

Manual Técnico

Este manual técnico tiene como objetivo describir las principales funciones de las clases con las que se implementó el sistema propuesto.

A.1 Class BrokerPlano

```
public class BrokerPlano
extends java.lang.Object
```

Esta clase provee del broker central necesario para la ejecución de aplicaciones distribuidas. El desarrollador no requiere sobre escribir ni implementar nada en esta clase. Código fuente en apéndice B.1

A.1.1 Clases Internas

BrokerPlano.lecturasocket Esta clase es la encargada de escuchar por el puerto asignado y esperar por comunicación proveniente de los clientes.

BrokerPlano.Receptor Esta clase se encuentra encargada de recibir nuevas conexiones de clientes, agregarlas al arreglo de conexiones y de colocarles en un sistemas de manejo que se encuentre escuchando dichas conexiones.

BrokerPlano.Shared Esta clase tiene por fin crear un sistema de *locks* para la escritura en las diferentes salidas, así solo un *thread* puede escribir a la vez por un mismo *socket*.

A.1.2 Constructor

BrokerPlano() El constructor se encarga de iniciar los arreglos de conexiones y los objetos internos para el manejo de conexiones.

A.1.3 Métodos Públicos

void iniciar() Este método se encarga de iniciar el servidor se *socket*s y de esperar por la comunicación de nuevos clientes.

static void main(java.lang.String Args) El método main solo se encarga de correr el broker interno cuando es llamado, no devuelve parámetros sobre su inicialización.

A.2 Class ClienteIntegral

```
public class ClienteIntegral
extends javax.swing.JApplet
```

Esta clase es el cliente de integración, se compone de una la parte visual (*applet*) y de una parte lógica de cálculo. La sección lógica de cálculo se encuentra dada por una clase que en este caso se llama "Integrador", la cual realiza todos los cálculos necesarios para retornar el resultados deseado. Código fuente en apéndice B.6

A.2.1 Clases Internas

ClienteIntegral.Integrador Esta clase esta encargada de realizar las peticiones de datos y los cálculos necesarios para obtener los resultados, que luego son enviados de regreso.

A.2.2 Métodos Públicos

void crearConexión() Este método crea la conexión (*socket*) hacia el servidor que en este caso es el broker. Ademas de eso inicia los canales de comunicación *DataInputStream* y *DataOutputStream*.

void init() Este es el método *init* del proceso de inicio de un *JApplet*, aquí se inician todos los elementos relacionados al GUI y algunos elementos de conexión como son el *hostname*. También inicia el proceso de conexión y comunicación con el coordinador. (esto significa que no requiere de acción extra por parte del usuario para iniciar)

A.3 Class ClienteMatriz

```
public class ClienteMatriz
extends javax.swing.JApplet
```

Esta clase es el cliente de multiplicación de matrices, se compone de una parte visual y una de lógica de cálculo. La sección de lógica de cálculo se encuentra dada por una clase interna de cálculo que en este caso se llama "Multiplicador". Código fuente en apéndice B.7.

A.3.1 Clases Internas

ClienteMatriz.Multiplicador Esta clase esta encargada de realizar las peticiones de datos y de realizar los cálculos necesarios para obtener los resultados que luego los envía de regreso.

A.3.2 Métodos Públicos

void crearConexión() Este método crea la conexión (*socket*) hacia el servidor que en este caso es el broker. Además de eso inicia los canales de comunicación *DataInputStream* y *DataOutputStream*.

void init() Este es el método *init* del proceso de inicio de un *JApplet*, aquí se inician todos los elementos relacionados al GUI y algunos elementos de conexión como son el *hostname*. También inicia el proceso de conexión y comunicación con el coordinador.

A.4 Class ClienteSORBroker

```
public class ClienteSORDirecto
extends javax.swing.JApplet
implements java.awt.event.ActionListener
```

Este Clase es el cliente para el cálculo de SOR para cuando tenemos comunicación por medio del broker.Código fuente en apéndice B.8.

A.4.1 Clases Internas

ClienteSORBroker.CalcularSOR Esta clase se encarga de realizar todos los procesos necesarios para realizar el cálculo, estos comprende desde la iniciación del servidor interno hasta el intercambio de mensajes entre los vecinos.

A.4.2 Métodos Públicos

actionPerformed(java.awt.event.ActionEvent evento) Este método llama a la creación de la conexión (*socket*) hacia el servidor que en este caso es el broker.

void crearConexión() Este método crea la conexión (*socket*) hacia el servidor que en este caso es el broker. Ademas de eso inicia los canales de comunicación *DataInputStream* y *DataOutputStream*.

void cerrarConexión() Este método se encarga de terminar la conexión, cerrando los *streams* y el *socket*.

void init() Este método se encarga de la inicialización de elementos de GUI, y ademas de ello llama a la creación de conexión y servidores y luego inicia el proceso interno de cálculo sin intervención del usuario.

A.5 Class ClienteSORDirecto

```
public class ClienteSORDirecto
extends javax.swing.JApplet
implements java.awt.event.ActionListener
```

Esta clase es el cliente para la ejecución de aplicación SOR con comunicación directa entre las partes. Código fuente en apéndice B.9.

A.5.1 Clases Internas

ClienteSORDirecto.cálculoSORDirecto Esta clase se encarga de realizar todos los procesos necesarios para realizar el cálculo, estos comprende desde la iniciación del servidor interno hasta el intercambio de mensajes entre los vecinos.

A.5.2 Métodos Públicos

actionPerformed(java.awt.event.ActionEvent evento) Este método captura las acciones del botón del GUI y realiza la acción de iniciar la aplicación de cálculo.

void init() Este método se encarga de la inicialización de elementos de GUI, y ademas de ello llama a la creación de conexión y servidores y luego inicia el proceso interno de cálculo sin intervención del usuario.

A.6 Class CoordinadorIntegral

```
public class CoordinadorIntegral
extends java.lang.Object
```

Esta es la clase del coordinador de la Integral, se encarga de todos los temas relacionados al proceso de coordinar los diversos clientes. Código fuente en apéndice B.2.

A.6.1 Constructor

CoordinadorIntegral(int num, *double* del,*double* limSup, *double* limInf) Constructor que se encarga de la inicialización de de parámetros para el coordinador. Recibe como entrada: un entero que es el numero de pedazos a dividirse la integral, un *double* delta que indica el pedazo de integracion para los clientes un *double* limite superior de la integral y un *double* limite inferior.

A.6.2 Métodos Públicos

void crearConexión() Este método se encarga de la creación de la conexión hacia el broker y recivir el id que le corresponde, en caso de no ser el id correcto (0) lo rechaza y termina el programa.

void cerrarConexión() Este método se encarga de cerrar la conexión con el broker de forma segura para la conexión, cerrando primero los *streams* y luego el *socket*.

void atender() Este método se encarga de todo el procedimiento de recepción y respuesta a peticiones por parte de los clientes. Al terminar el trabajo informa el tiempo tomado en ejecutarlo.

static void main(java.lang.String Args) Este es el método central que se encarga de recibir los parámetros dados por el usuario, y en caso de ser necesario desplegar una pequeña ayuda. Cuando los datos ingresados son correctos crea, inicia y ejecuta el CoordinadorIntegral para que este realice su trabajo.

A.7 Class CoordinadorMatriz

```
public class CoordinadorMatriz
extends java.lang.Object
```

Esta es la clase del coordinador de la Multiplicación de Matrices, se encarga de todos los temas relacionados al proceso de coordinar los diversos clientes. Código fuente en apéndice B.3.

A.7.1 Constructor

CoordinadorMatriz(int ejeAx, int ejeAy, int ejeBx, int ejeBy) Este es el constructor que se encarga de elementos de inicialización de la clase. Toma como entradas cuatro enteros, los que representan en pares los ejes de las matrices de A y B, primero A de la siguiente forma: A(x,y) y luego B (x,y). Por el momento no existe un método para el ingreso de matrices.

A.7.2 Métodos Públicos

void crearConexión() Este método se encarga de la creación de la conexión hacia el broker y recibir el id que le corresponde, en caso de no ser el id correcto (0) lo rechaza y termina el programa.

void cerrarConexión() Este método se encarga de cerrar la conexión con el broker de forma segura para la conexión, cerrando primero los *streams* y luego el *socket*.

void atender() Este método se encarga de todo el procedimiento de recepción y respuesta a peticiones por parte de los clientes. Al terminar el trabajo informa el tiempo tomado en ejecutarlo.

static void main(java.lang.String Args) Este es el método central que se encarga de recibir los parámetros dados por el usuario, y en caso de ser necesario desplegar una pequeña ayuda. Cuando los datos ingresados son correctos crea, inicia y ejecuta el CoordinadorIntegral para que este realice su trabajo.

A.8 Class CoordinadorSORBroker

```
public class CoordinadorSORBroker
extends java.lang.Object
```

Esta es la clase del coordinador de SOR con broker, se encarga de todos los temas relacionados al proceso de coordinar los diversos clientes. Código fuente en apéndice B.4.

A.8.1 Constructor

CoordinadorSORBroker(byte idMin, byte idMax, int inte, int X, int Y) El constructor se encarga de tomar los parámetros para el programa y realizar la inicialización de las variables internas. Toma como parámetros un byte indicando el id mínimo, otro byte indicando el id máximo, un entero con el número de interacción y dos enteros mas con los ejes X y Y.

A.8.2 Métodos Públicos

void crearConexión() Este método se encarga de la creación de la conexión hacia el broker y recibir el id que le corresponde, en caso de no ser el id correcto (0) lo rechaza y termina el programa.

void cerrarConexión() Este método se encarga de cerrar la conexión con el broker de forma segura para la conexión, cerrando primero los *streams* y luego el *socket*.

void atender() Este método se encarga de todo el procedimiento de recepción y respuesta a peticiones por parte de los clientes. Al terminar el trabajo informa el tiempo tomado en ejecutarlo.

static void main(java.lang.String Args) Este es el método central que se encarga de recibir los parámetros dados por el usuario, y en caso de ser necesario desplegar una pequeña ayuda. Cuando los datos ingresados son correctos crea, inicia y ejecuta el CoordinadorIntegral para que este realice su trabajo.

A.9 Class CoordinadorSORDirecto

```
public class CoordinadorSORDirecto
extends java.lang.Object
```

Esta es la clase del coordinador de SOR con comunicación directa, se encarga de todos los temas relacionados al proceso de coordinar los diversos clientes. Código fuente en apéndice B.5.

A.9.1 Constructor

CoordinadorSORDirecto(byte idMin, byte idMax, int inter, int X, int Y) El constructor se encarga de tomar los parámetros para el programa y realizar la inicialización de las variables internas. Toma como parámetros un byte indicando el id mínimo, otro byte indicando el id máximo, un entero con el número de interacción y dos enteros mas con los ejes X y Y.

A.9.2 Métodos Públicos

void crearConexión() Este método se encarga de la creación de la conexión hacia el broker y recibir el id que le corresponde, en caso de no ser el id correcto (0) lo rechaza y termina el programa.

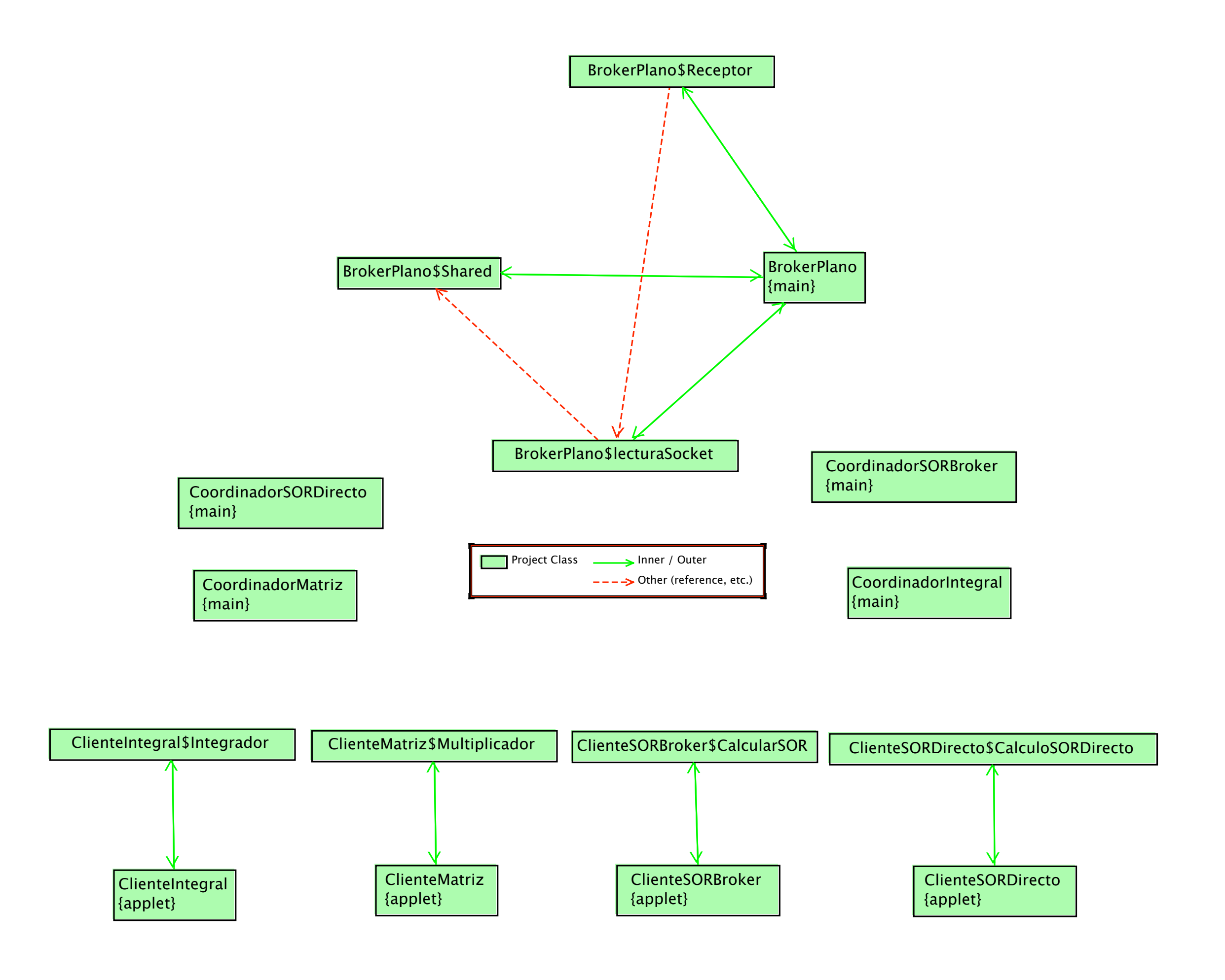


Figura A.1: Diagrama de Clases.

void cerrarConexión() Este método se encarga de cerrar la conexión con el broker de forma segura para la conexión, cerrando primero los *streams* y luego el *socket*.

void atender() Este método se encarga de todo el procedimiento de recepción y respuesta a peticiones por parte de los clientes. Al terminar el trabajo informa el tiempo tomado en ejecutarlo.

static void main(java.lang.String Args) Este es el método central que se encarga de recibir los parámetros dados por el usuario, y en caso de ser necesario desplegar una pequeña ayuda. Cuando los datos ingresados son correctos crea, inicia y ejecuta el CoordinadorIntegral para que este realice su trabajo.

Apéndice B

Código Fuente de las Aplicaciones Reales

B.1 *Broker* Genérico

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.io.DataInputStream;
```

```
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
/*
 * Esta es la implementacion del Broker generico
 */
/**
 * Esta clase provee del broker central necesario para
 * la ejecución de aplicaciones distribuidas. El desarrollador
 * no requiere sobre escribir ni implementar nada en esta clase.
 */
public class BrokerPlano {
//elementos de conexion
private final int sizeBuffer = 4096;
private Socket[] arregloConexiones;
private DataInputStream[] entradas;
private DataOutputStream[] salidas;
private lecturaSocket[] HiloSockets;
private ServerSocket servidor;
private int port = 4545;
//elemtos de bloqueo compartido
volatile Shared criticalSection = new Shared();
//medicion de tiempo
private long tiempoIncial;
private long tiempoFinal;
//Objeto de recepcion
private Receptor recepcion;

/**
 * El constructor se encarga de incializar los arreglos de conexiones
 * y los objetos internos para el manejo de conexiones.
 */

public BrokerPlano() {
         arregloConexiones = new Socket[256];
        entradas = new DataInputStream[256];
         salidas = new DataOutputStream[256];
         HiloSockets = new lecturaSocket[256];
         recepcion = new Receptor();

     }
/**
 * Este metodo se encarga de iniciar el servidor se sockets
 * y de comenzar la escucha de clientes.
 */
public void iniciar() {
recepcion.iniciar();
         recepcion.start();
}

/**
 * Esta clase se encuentra encargada de recivir nuevas conexiones
 * de nuevos clientes y agregarlas al arreglo de conexiones ademas
 * de colocarles un manejador que se encuentre escuchando dicha conexion.
```

```
 */
 public class Receptor extends Thread {
/**
 * Este metodo tiene por objetivo crear el nuevo servidor de
 * sockets "ServerSocket" y dejarlo escuchando.
 */
        public void iniciar() {
            try {
                servidor = new ServerSocket(port);
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }

/**
 * Este metodo ejecutado por el thread, aqui es donde se crean las nuevas conexiones,
 * se las identifica y se les asigna un "LecturaSocket" para que las supervise.
 */
        public void run() {
            int puertoInstanciar = 0;
            try {
                while (puertoInstanciar < 256) {
                    arregloConexiones[puertoInstanciar] = servidor.accept();
                    arregloConexiones[puertoInstanciar].setSendBufferSize(131072);
                    arregloConexiones[puertoInstanciar].setReceiveBufferSize(131072);
                    // System.out.println("Conexion "+puertoInstanciar);
                    entradas[puertoInstanciar] =
                        new DataInputStream(arregloConexiones[puertoInstanciar]
                            .getInputStream());
                    salidas[puertoInstanciar] =
                        new DataOutputStream(arregloConexiones[puertoInstanciar]
                            .getOutputStream());
                    //para el caso de Matricial le envio su identificador
                    salidas[puertoInstanciar].writeByte((byte) puertoInstanciar);

                    //creo el tread y le digo cual escuchar
                    HiloSockets[puertoInstanciar] = new lecturaSocket(puertoInstanciar);
                    HiloSockets[puertoInstanciar].start();
                    puertoInstanciar++;
                }
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }
    }// fin clase receptora

/**
 * Esta clase es la encargada de escuchar por su puerto
 * asignado y esperar por comunicacion proveniente de los
 * clientes.
 */
public class lecturaSocket extends Thread {
        private int puertoDondeLeo;
        public byte[] buffer = new byte[sizeBuffer];
```

```
/**
 * El constructor recive como parametros un entero que le indica
 * que puerto debe leer dentro del arreglo de puertos establecido,
 * indicandole de cual puerto esta encargado.
 */
        public lecturaSocket(int donde ) {
            puertoDondeLeo = donde;
        }
        /**
 * Este metodo tiene por objetivo cerrar la conexion asignada a este
 * thread, este cierre se lo realiza de forma correcta sin forzar la conexion.
 */
       public void cerrar() {
           try {
               entradas[puertoDondeLeo].close();
               salidas[puertoDondeLeo].close();
               arregloConexiones[puertoDondeLeo].close();
           } catch (Exception e) {
               System.out.println("problemas cerrando");
               e.printStackTrace();
           }
       }
        /**
 * Este metodo es el ejecutado por el thread directamente y aqui es donde se
 * encuentra le proceso por el cual cuando llegan datos son conmutados hacia el
 * destinatario. Este proceso necesita la toma de posecion del lock para poder
 * realizar la escritura al socket que desea.
 */
        public void run() {
try {
                  while(true) {
                      int salida = entradas[puertoDondeLeo].readByte();
                      int numeroLecturas = entradas[puertoDondeLeo].readInt();
                      synchronized (criticalSection) {
                          while (criticalSection.getLock() == false)
                          sleep(1);
                           //System.out.println(puertoDondeLeo + " -> " +
                          //                   salida  +  " " + numeroLecturas);
                         salidas[salida].writeByte(puertoDondeLeo);
                         salidas[salida].writeInt(numeroLecturas);
                         while(numeroLecturas > 0) {
                             int restar = entradas[puertoDondeLeo].read(buffer,
                                                     0, Math.min(numeroLecturas, sizeBuffer));
                              salidas[salida].write(buffer, 0, restar);
                              numeroLecturas -= restar;
                         }
                         if (criticalSection.releaseLock() == false)
                             System.out.println("Error Lock");
                     }
 }
             } catch(EOFException e) {
                 // System.out.println("EOF");
                 this.cerrar();
             } catch (IOException e) {
             System.out.println("ERROR:IOE");
```

```
                this.cerrar();
        } catch (Exception e) {
                e.printStackTrace();
            }
         }
     }//fin clase lecturaSocket

/**
 * Esta clase tiene por fin crear un sistema de locks para la escritura
 * en las diferentes salidas, asi solo un thread puede escribir a la vez
 * por un mismo socket.
 */
public class Shared {
            private boolean lock;

Shared () {
lock = false;
}
/**
 * Metodo para la obtencion del lock,
 * retorna true en caso de exito.
 */
public synchronized boolean getLock() {
if (lock == true)
return false;
lock = true;
return true;
}
/**
 * Metodo para la liberacion del lock,
 * retorna true en caso de exito.
 */
public synchronized boolean releaseLock() {
if (lock == false)
return false;
lock = false;
return true;
}
}
/**
 * Solo esta encargado de lanzar el broker interno cuando es
 * llamado, no indica ningun estado de regreso.
 */
public static void main (String Args[]) {
        BrokerPlano broker = new BrokerPlano();
broker.iniciar();
}
}
```

B.2 Coordinador de la Integración Numérica

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
```

```
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;
/**
 * Esta es la clase del coordinador de la Integral,
 * se encarga de todos los temas relacionados al proceso
 * de coordinar los diversos clientes.
 */
public class CoordinadorIntegral {
    private double Integral = 0; //resultado de la integral
    private double limInferior = 0; //limite inferior
    private double limSuperior = 10; //limite superior de la integral
    private double delta = 0.00000001; //delta de la integral
    private int numero; //numero de pedazos a dividir

    private Socket conexion;
    private DataInputStream entradaDatos;
    private DataOutputStream salidaDatos;
    private String host = "localhost";
    private int port = 4545;
    private byte id;

    private byte idDevuelta;
```

```
    private byte comando;
    //para medir tiempos
    private long tiempoInicio;
    private long tiempoFinal;

/**
 * Constructor que se encarga de la inicializacion de
 * de parametros para el coordinador. Recive como entrada:
 * un entero que es el numero de pedazos a dividirse la integral,
 * un double delta que indica el pedazo de integracion para los clientes
 * un double limite superior de la integral y un double limite inferior.
 */
    public CoordinadorIntegral(int numero, double delta,
                            double limiteSuperior, double limiteInferior ) {
        this.delta = delta;
        this.numero = numero;
        this.limInferior = limiteInferior;
        this.limSuperior = limiteSuperior;
    }
    /**
 * Este metodo se encarga de la creacion de la conexion hacia el
 * el broker y recivir el id que le corresponde, en caso de no ser
 * el id correcto (0) lo rechaza y termina el programa
 */
    public void crearConexion() throws Exception {
        conexion = new Socket(host, port);
        entradaDatos = new DataInputStream(conexion.getInputStream());
        salidaDatos = new DataOutputStream(conexion.getOutputStream());
        id = entradaDatos.readByte();
        if (id != 0) {
            System.out.println("ERROR: El id no es 0");
            System.exit(1);
        }
        //System.out.println("Configurador Instanciado (" + id + ")");
    }
    /**
 * Este metodo se encarga de cerra la conexion con el broker
 * de forma segura para la conexion, cerrando primero los
 * streams y luego el socket.
 */
    public void cerrarConexion() throws Exception {
        entradaDatos.close();
        salidaDatos.close();
        conexion.close();
    }
    /**
 * Este metodo se encarga de todo el procedimiento de recepcion
 * y respuesta a peticiones por parte de los clientes. Al terminar
 * el trabajo informa el tiempo tomado en ejecutarlo.
 */
    public void atender() {
        //este metodo respondera a los pedidos realizados
        boolean porEntregar = true;
        int numeroTerminados = 0;
        int numeroEnviados = 0;
        boolean tiempoInicial = false;
```

```
        while(numeroTerminados < numero) {
            try {
                //primero el id para devolver los parametros
                idDevuelta = entradaDatos.readByte();
                int size = entradaDatos.readInt();
                if (tiempoInicial == false) {
                    tiempoInicio = System.currentTimeMillis();
                    tiempoInicial = true;
                }
                comando = entradaDatos.readByte();
                if (comando == (byte) 1) {
                    // System.out.println("Pidiendo los datos  para multiplicar");
                    if (porEntregar == false) {
                        salidaDatos.writeByte(idDevuelta); //a quien envio el mensaje
                        salidaDatos.writeInt(0); //conmando 0
                    }
                    else {
                        double inicio = limInferior +
                            (limSuperior - limInferior)/numero*numeroEnviados;
                        salidaDatos.writeByte(idDevuelta); //a quien envio el mensaje
                        salidaDatos.writeInt(3*8); //envio 3 doubles (tam mensaje)
                        salidaDatos.writeDouble(inicio); //limite inferior y superior
                        salidaDatos.writeDouble(inicio + (limSuperior - limInferior)/numero);
                        salidaDatos.writeDouble(delta);
                        numeroEnviados++;
                        if (numeroEnviados == numero) {
                            porEntregar = false;
                        }
                    }
                }
                else if (comando == (byte) 2) {
                    Integral += entradaDatos.readDouble();
                    numeroTerminados++;
                }
                //System.out.println("Terminada la operacion");
            } catch (Exception e){
                e.printStackTrace();
                System.exit(1);
            }
        }
        tiempoFinal = System.currentTimeMillis();
        //System.out.println("terminado !!");
        System.out.println("El tiempo fue de " + (tiempoFinal-tiempoInicio)
                           + " milisegundos");
    }
    /**
 * Este es el metodo central que se encarga de recivir los parametros
 * dados por el usuario, y en caso de ser necesario desplegar una pequena
 * ayuda. Cuando los datos ingresados son correctos crea, inicializa y lanza
 * al CoordinadorIntegral para que este realice su trabajo.
 */
    public static void main(String Args[]){
        if (Args.length < 4){
            System.out.println("Uso: java ConfiguradorIntegral <# pedazos> <delta>" +
                               " <limite Superior> <limite Inferior>");
            System.exit(0);
```

```
        }

        CoordinadorIntegral xx = new CoordinadorIntegral(Integer.parseInt(Args[0]),
                                                         Double.parseDouble(Args[1]),
                                                         Double.parseDouble(Args[2]),
                                                         Double.parseDouble(Args[3]));
        try {
            xx.crearConexion();
        } catch (Exception e){
            e.printStackTrace();
            System.exit(1);
        }
        xx.atender();
    }
}
```

B.3 Coordinador de la Multiplicación de Matrices

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.io.DataInputStream;
import java.io.DataOutputStream;
```

```
import java.net.Socket;

public class CoordinadorMatriz {
    //esta es la applicacion que se encuentra escuchaod en el puerto 0 del Broker
    //tamano de la matriz que se va multiplicar A
    private int ejeAX = 10000;
    private int ejeAY = 12;
    //matriz B
    private int ejeBX = 12;
    private int ejeBY = 10000;
    //Almacenamiento de las matrices
    private double[][] matrizA;
    private double[][] matrizB;
    private double[][] matrizRespuesta;
    private int barridoA = 0;
    private int barridoB = 0;
    private int numeroDeBytesPorDoble = 8;
    //elementos de conexion
    private Socket conexion;
    private DataInputStream entradaDatos;
    private DataOutputStream salidaDatos;
    private String host = "localhost";
    private int port = 4545;
    private byte id;

    private byte idDevuelta;
    private byte comando;

    //para medicion de tiempos
    private long tiempoInicio;
    private long tiempoFinal;

/**
 * Este es el constructor que se encarga de elementos de inicializacion
 * de la clase. Toma como entradas cuatro enteros, los que representan en
 * pares los ejes de las matrices de A y B, primero A de la siguiente forma:
 * A(x,y) y luego B (x,y). Por el momento no existe un metodo para el ingreso
 * de matrices.
 */
    public CoordinadorMatriz (int ejeAx, int ejeAy, int ejeBx, int ejeBy ){
        //primero configuramos los datos
        this.ejeAX = ejeAx;
        this.ejeAY = ejeAY;
        this.ejeBX = ejeBx;
        this.ejeBY = ejeBy;

        //creamos las matrices
        if (ejeAX != ejeBY) {
            System.out.println("ERROR: las dimensiones no son las correctas " +
                                   ejeAX + " " + ejeBY );
            System.exit(1);
        }
        matrizA = new double[ejeAY][ejeAX];
        matrizB = new double[ejeBY][ejeBX];
        matrizRespuesta = new double[ejeAY][ejeBX];
```

```
        //rellenamos de los datos (falsos)
        for (int i = 0; i < ejeAY; i++) {
            for(int j = 0 ; j < ejeAX; j++) {
                matrizA[i][j] = 1.0;
            }
        }
        for (int i = 0; i < ejeBY; i++) {
            for(int j = 0 ; j < ejeBX; j++) {
                matrizB[i][j] = 1.0;
            }
        }
        for (int i = 0; i < ejeAY; i++) {
            for(int j = 0 ; j < ejeBX; j++) {
                matrizRespuesta[i][j] = -1.0;
            }
        }
    }
    /**
 * Este metodo se encarga de la creacion de la conexion hacia el
 * el broker y recivir el id que le corresponde, en caso de no ser
 * el id correcto (0) lo rechaza y termina el programa
 */
    public void crearConexion() throws Exception {
        conexion = new Socket(host, port);
        entradaDatos = new DataInputStream(conexion.getInputStream());
        salidaDatos = new DataOutputStream(conexion.getOutputStream());
        id = entradaDatos.readByte();
        if (id != 0) {
            System.out.println("ERROR: id diferente de 0");
            System.exit(0);
        }
        //System.out.println("Configurador Instanciado ("+id+")");
    }
    /**
 * Este metodo se encarga de cerra la conexion con el broker
 * de forma segura para la conexion, cerrando primero los
 * streams y luego el socket.
 */
    public void cerrarConexion() throws Exception {
        entradaDatos.close();
        salidaDatos.close();
        conexion.close();
    }
    /**
 * Este metodo se encarga de todo el procedimiento de recepcion
 * y respuesta a peticiones por parte de los clientes. Al terminar
 * el trabajo informa el tiempo tomado en ejecutarlo.
 */
    public void atender(){
        //este metodo respondera a los pedidos realizados
        boolean porEntregar = true;
        int numeroTerminados = 0;
        boolean tiempoInicial = false;
        while(numeroTerminados < ejeAY*ejeBX) {
            try {
                //primero el id
```

```
        idDevuelta = entradaDatos.readByte();
        int size = entradaDatos.readInt();
        if (tiempoInicial == false) {
            tiempoInicio = System.currentTimeMillis();
            tiempoInicial = true;
        }
        /*luego que desea (byte); 0 para configurar, 1 para nueva fila,
          2 para devolver resultados*/
        comando = entradaDatos.readByte(); //instruccion que desea ejecutar
        if (comando == (byte) 0) {
            salidaDatos.writeByte(idDevuelta); //a quien envio el mensaje
            salidaDatos.writeInt(4);
            salidaDatos.writeInt(ejeAX);
        }
        else if (comando == (byte) 1) {
            // System.out.println("Pidiendo los datos para multiplicar");
            if (porEntregar == false) {
                salidaDatos.writeByte(idDevuelta); //a quien envio el mensaje
                salidaDatos.writeInt(0);
            }
            else {
                salidaDatos.writeByte(idDevuelta); //a quien envio el mensaje
                salidaDatos.writeInt(2*ejeAX*numeroDeBytesPorDoble+2*4);
                //siempre primero A, luego B;
                salidaDatos.writeInt(barridoA);
                salidaDatos.writeInt(barridoB);
                //ahora si los datos en el mismo orden
                for(int i = 0; i < ejeAX; i++){
                    salidaDatos.writeDouble(matrizA[barridoA][i]);
                }
                for(int i = 0; i < ejeBY; i++){
                    salidaDatos.writeDouble(matrizB[i][barridoB]);
                }
                //terminada la operacion pedida, actualizar datos
                barridoB++;
                if(barridoB == ejeBX){
                    barridoB = 0;
                    barridoA++;
                }
                if(barridoA == ejeAY){
                    //ambos llegaron a su maximo, no quedan mas linea a entregar
                    porEntregar = false;
                }
            }
        }
        else if (comando == (byte) 2) {
            //System.out.println("Devolviendo resultados");
            //primero la posicion
            int posicionA = entradaDatos.readInt();
            int posicionB = entradaDatos.readInt();
            matrizRespuesta[posicionA][posicionB] = entradaDatos.readDouble();
            numeroTerminados++;
        }
        //System.out.println("Terminada la operacion");
    } catch (Exception e){
        e.printStackTrace();
```

```
                System.exit(1);
            }
        }
        tiempoFinal = System.currentTimeMillis();
        System.out.println("El tiempo fue de " + (tiempoFinal-tiempoInicio)
                          + " milisegundos");
        //System.out.println(matrizRespuesta[2][2]); //para comprobar el resultados
    }
    /**
 * Este es el metodo central que se encarga de recivir los parametros
 * dados por el usuario, y en caso de ser necesario desplegar una pequena
 * ayuda. Cuando los datos ingresados son correctos crea, inicializa y lanza
 * al CoordinadorIntegral para que este realice su trabajo.
 */
    public static void main(String Args[]){
        if (Args.length < 4){
            System.out.println("Uso: java ConfiguradorMatriz <ejeAX> <ejeAY> <ejeBX>" +
  " <ejeBY>");
            System.exit(0);
        }
        CoordinadorMatriz xx = new CoordinadorMatriz(Integer.parseInt(Args[0]),
                                                     Integer.parseInt(Args[1]),
                                                     Integer.parseInt(Args[2]),
                                                     Integer.parseInt(Args[3]));
        try {
            xx.crearConexion();
        } catch (Exception e){
            e.printStackTrace();
            System.exit(1);
        }
        xx.atender();
    }
}
```

B.4 Coordinador de SOR con Comunicación Indirecta

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
```

```
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;
/**
 *
 */
public class CoordinadorSORBroker {
//elementos de configuracion
    private int numeroInteracciones;
    private int ejeX = 1500;
    private int ejeY = 4800; //para cuatro solo cliente
    private byte idMinimo;
    private byte idMaximo;
//elementos estandar de bytes
private int numeroDeBytesPorDoble = 8;
    private int numeroDeBytesPorEntero = 4;
//Elementos de conexion
private Socket conexion;
private DataInputStream entradaDatos;
private DataOutputStream salidaDatos;
private String host = "localhost";
private int port = 4545;
private byte id;
private byte idDevuelta;
private byte comando;
//tiempos
private long tiempoInicio;
private long tiempoFinal;
    /**
 * El constructor se encarga de tomar los parametros para el programa y
 * realizar la inicializacion de las varaibles internas. Toma como parametros
 * un byte indicandole el id minimo, otro byte indicandole el id maximo, un entero
 * con el numero de interaccion y dos enteros mas con los ejes X y Y.
 */
    public CoordinadorSORBroker(byte idMin, byte idMax, int interaciones,
                   int ejeX, int ejeY) {
        idMinimo = idMin;
        idMaximo = idMax;
```

```
        numeroInteracciones = interaciones;
        this.ejeX = ejeX;
        this.ejeY = ejeY;
    }
    /**
* Este metodo se encarga de la creacion de la conexion hacia el
* el broker y recivir el id que le corresponde, en caso de no ser
* el id correcto (0) lo rechaza y termina el programa
*/
    public void crearConexion() throws Exception{
        conexion = new Socket(host,port);
        entradaDatos = new DataInputStream(conexion.getInputStream());
        salidaDatos = new DataOutputStream(conexion.getOutputStream());
        id = entradaDatos.readByte();
        if (id != 0)
            System.out.println("ERROR");
        // System.out.println("Configurador Instanciado ("+id+")");
    }
    /**
* Este metodo se encarga de cerra la conexion con el broker
* de forma segura para la conexion, cerrando primero los
* streams y luego el socket.
*/
    public void cerrarConexion() throws Exception {
        entradaDatos.close();
        salidaDatos.close();
        conexion.close();
    }
    /**
* Este metodo se encarga de todo el procedimiento de recepcion
* y respuesta a peticiones por parte de los clientes. Al terminar
* el trabajo informa el tiempo tomado en ejecutarlo.
*/
    public void atender(){
        double matriz[][] = new double[ejeY][ejeX];
        //este metodo respondera a los pedidos realizados
        boolean porEntregar = true;
        boolean continuarLazo = true;
        boolean tiempoInicial = false;
        //boolean tiempoFin = false;
        int numeroRetornos = idMaximo - idMinimo + 1;
        int numeroLecturas;

        for (int i = 0; i < ejeY; i++)
            for (int j = 0; j < ejeX; j++)
                matriz[i][j] = 1;

        int sizeY = ejeY/(idMaximo - idMinimo + 1);
        while (continuarLazo) {
            try {
                //primero el id
                idDevuelta = entradaDatos.readByte();
                numeroLecturas = entradaDatos.readInt();
                //System.out.println("Rcliente " + numeroLecturas + " viene de " + idDevuelta);
                if (tiempoInicial == false) {
                    tiempoInicio = System.currentTimeMillis();
```

```
                tiempoInicial = true;
            }
            //luego que desea (byte); 1 para nueva fila, 2 para devolver resultados
            comando = entradaDatos.readByte();
            //System.out.println("comando " + comando );
            if (comando == (byte) 1) {
                if (porEntregar == false) {
                    salidaDatos.writeByte(idDevuelta);
                    salidaDatos.writeInt(0);
                }
                else {
                    int sizeCambiado = sizeY;
                    if (idDevuelta != idMinimo)
                        sizeCambiado++;
                    if (idDevuelta != idMaximo)
                        sizeCambiado++;
                    /*System.out.println("EC" +( 2 + 3*numeroDeBytesPorEntero +
                      (ejeX*sizeCambiado*numeroDeBytesPorDoble)));*/
                    salidaDatos.writeByte(idDevuelta);
                    salidaDatos.writeInt(2 + 3*numeroDeBytesPorEntero +
                                         (ejeX*sizeCambiado*numeroDeBytesPorDoble));
                    salidaDatos.writeByte(idMaximo);
                    salidaDatos.writeByte(idMinimo);
                    salidaDatos.writeInt(ejeX);
                    salidaDatos.writeInt(sizeCambiado);
                    salidaDatos.writeInt(numeroInteracciones);

                    int posInicio = (idDevuelta - idMinimo)*sizeY;
                    if (idDevuelta != idMinimo)
                        posInicio--;
                    for (int i = 0; i < sizeCambiado; i++)
                        for (int j = 0; j < ejeX; j++)
                            salidaDatos.writeDouble(matriz[posInicio + i][j]);

                }
            }
            else if (comando == (byte) 2) {
                int posInicio = (idDevuelta - idMinimo)*sizeY;
                //System.out.println("Nlecturas " + ejeX*sizeY*8);
                for (int i = 0; i < sizeY; i++)
                    for (int j = 0; j < ejeX; j++)
                        matriz[posInicio + i][j] = entradaDatos.readDouble();
                numeroRetornos--;
                if (numeroRetornos == 0) {
                    tiempoFinal = System.currentTimeMillis();
                    continuarLazo = false;
                    //System.out.println("Terminada Todo");
                }
                //System.out.println("Terminada la operacion");
            }
            else {
                System.exit(1);
            }
        } catch (Exception e){
            e.printStackTrace();
            System.exit(1);
```

```
            }
        }
        System.out.println("El tiempo fue de "+ (tiempoFinal-tiempoInicio) +
                           " milisegundos con " + idMaximo + " de cliente ");
    }
    /**
 * Este es el metodo central que se encarga de recivir los parametros
 * dados por el usuario, y en caso de ser necesario desplegar una pequena
 * ayuda. Cuando los datos ingresados son correctos crea, inicializa y lanza
 * al CoordinadorIntegral para que este realice su trabajo.
 */
    public static void main(String Args[]){
        if (Args.length < 5){
            System.out.println("Modo de Uso: Java Coordinador <idMinimo>" +
                               " <idMaximo> <Numero Interacciones>" +
                               " <EjeX> <Ejey>");
            System.exit(0);
        }
        CoordinadorSORBroker xx = new CoordinadorSORBroker(Byte.parseByte(Args[0]),
                                         Byte.parseByte(Args[1]),
                                         Integer.parseInt(Args[2]),
                                         Integer.parseInt(Args[3]),
                                         Integer.parseInt(Args[4]));
        try {
            xx.crearConexion();
        } catch (Exception e){
            e.printStackTrace();
            System.exit(1);
        }
        xx.atender();
    }

}
```

B.5 Coordinador de SOR con Comunicación Directa

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
```

```
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.InetAddress;
import java.net.Socket;
/**
 * Este es la clase que se encuarga de la coordinacion
 * de los clientes para la aplicacion de SOR por comunicacion
 * directa.
 */
public class CoordinadorSORDirecto {
//parametros de configuracion
    private int numeroInteracciones;
    private int ejeX = 1500;
    private int ejeY = 4800;
    private byte idMinimo;
    private byte idMaximo;
    //Elementos no variables de bytes
    private int numeroDeBytesPorDoble = 8;
    private int numeroDeBytesPorEntero = 4;
    //Elementos de conexion
    private Socket conexion;
    private DataInputStream entradaDatos;
    private DataOutputStream salidaDatos;
    private String host = "localhost";
    private int port = 4545;
    private byte id;
    private byte idDevuelta;
    private byte comando;
    //para los keyId de cada cliente
    private String[] keyId;
    //tiempos
    private long tiempoInicio;
    private long tiempoFinal;
    /**
 * El constructor se encarga de tomar los parametros para el programa y
 * realizar la inicializacion de las varaibles internas. Toma como parametros
 * un byte indicandole el id minimo, otro byte indicandole el id maximo, un entero
 * con el numero de interaccion y dos enteros mas con los ejes X y Y.
```

```
*/
   public CoordinadorSORDirecto (byte idMin, byte idMax, int interaciones,
                              int ejeX, int ejeY) {
       idMinimo = idMin;
       idMaximo = idMax;
       numeroInteracciones = interaciones;
       this.ejeX = ejeX;
       this.ejeY = ejeY;
       keyId = new String[idMaximo + 1];
   }
   /**
* Este metodo se encarga de la creacion de la conexion hacia el
* el broker y recivir el id que le corresponde, en caso de no ser
* el id correcto (0) lo rechaza y termina el programa
*/
   public void crearConexion() throws Exception {
       conexion = new Socket(host,port);
       entradaDatos = new DataInputStream(conexion.getInputStream());
       salidaDatos = new DataOutputStream(conexion.getOutputStream());
       id = entradaDatos.readByte();
       if (id != 0) {
           System.out.println("ERROR: id es diferente de 0");
           System.exit(1);
       }
       //System.out.println("Configurador Instanciado ("+id+")");
   }
   /**
* Este metodo se encarga de cerra la conexion con el broker
* de forma segura para la conexion, cerrando primero los
* streams y luego el socket.
*/
   public void cerrarConexion() throws Exception {
       entradaDatos.close();
       salidaDatos.close();
       conexion.close();
   }
   /**
* Este metodo se encarga de todo el procedimiento de recepcion
* y respuesta a peticiones por parte de los clientes. Al terminar
* el trabajo informa el tiempo tomado en ejecutarlo.
*/
   public void atender() {
       double matriz[][] = new double[ejeY][ejeX];
       boolean porEntregar = true;
       boolean continuarLazo = true;
       boolean tiempoInicial = false;
       //boolean tiempoFin = false;
       int numeroRetornos = idMaximo - idMinimo + 1;
       int numeroLecturas;

       for (int i = 0; i < ejeY; i++)
           for (int j = 0; j < ejeX; j++)
               matriz[i][j] = 1;

       int sizeY = ejeY/(idMaximo - idMinimo + 1);
       while (continuarLazo) {
```

```
        try {
            //primero el id
            idDevuelta = entradaDatos.readByte();
            numeroLecturas = entradaDatos.readInt();
            //System.out.println("Rcliente " + numeroLecturas + " viene de " + idDevuelta);
            if (tiempoInicial == false) {
                tiempoInicio = System.currentTimeMillis();
                tiempoInicial = true;
            }
            /* luego que desea (byte); 1 para nueva fila, 2 para devolver resultados
             * y comando 3 para enviar su id address (bytes) hacia aca, luego espera que le
             * le devuelvan el ip del que debe conectarse
             */
            //System.out.println("comando " + comando );
            comando = entradaDatos.readByte();
            if (comando == (byte) 1) {
                if (porEntregar == false) {
                    salidaDatos.writeByte(idDevuelta);
                    salidaDatos.writeInt(0);
                }
                else {
                    int sizeCambiado = sizeY;
                    if (idDevuelta != idMinimo)
                        sizeCambiado++;
                    if (idDevuelta != idMaximo)
                        sizeCambiado++;
                    /*System.out.println("EC" +( 2 + 3*numeroDeBytesPorEntero +
                      (ejeX*sizeCambiado*numeroDeBytesPorDoble)));*/
                    salidaDatos.writeByte(idDevuelta);
                    salidaDatos.writeInt(2 + 3*numeroDeBytesPorEntero +
                                         (ejeX*sizeCambiado*numeroDeBytesPorDoble));
                    salidaDatos.writeByte(idMaximo);
                    salidaDatos.writeByte(idMinimo);
                    salidaDatos.writeInt(ejeX);
                    salidaDatos.writeInt(sizeCambiado);
                    salidaDatos.writeInt(numeroInteracciones);

                    int posInicio = (idDevuelta - idMinimo)*sizeY;
                    if (idDevuelta != idMinimo)
                        posInicio--;
                    for (int i = 0; i < sizeCambiado; i++)
                        for (int j = 0; j < ejeX; j++)
                            salidaDatos.writeDouble(matriz[posInicio + i][j]);
                }
            }
            else if (comando == (byte) 2) {
                int posInicio = (idDevuelta - idMinimo)*sizeY;
                //System.out.println("Nlecturas " + ejeX*sizeY*8);
                for (int i = 0; i < sizeY; i++)
                    for (int j = 0; j < ejeX; j++)
                        matriz[posInicio + i][j] = entradaDatos.readDouble();
                numeroRetornos--;
                if (numeroRetornos == 0) {
                    tiempoFinal = System.currentTimeMillis();
                    continuarLazo = false;
                    //System.out.println("Terminada Todo");
```

```
                }
                //System.out.println("Terminada la operacion");
            } else if (comando == (byte) 3) {
                //recivir el ip addres del que se conecto
                byte[] IpByte = new byte[4];
                entradaDatos.readFully(IpByte);
                String hostIp = InetAddress.getByAddress(IpByte).getHostAddress();
                keyId[idDevuelta] = hostIp;
                //System.out.println("obtenida la direccion " + hostIp);
            } else if (comando == (byte) 4) {
                //devolver el ip del siguiente si lo tengo.
                if (keyId[idDevuelta + 1] == null) {
                    //no lo tengo todavia
                    salidaDatos.writeByte(idDevuelta);
                    salidaDatos.writeInt(0); //no esta listo
                } else {
                    //lo tengo listo
                    salidaDatos.writeByte(idDevuelta);
                    salidaDatos.writeInt(4); //son 4 bytes de dir (IPv4)
                    salidaDatos.write(InetAddress.getByName(keyId[idDevuelta + 1]).
                                        getAddress());
                }
            }
            else {
                System.out.println("ERROR: comando desconocido " + comando);
                System.exit(1);
            }
        } catch (Exception e){
            e.printStackTrace();
            System.exit(1);
        }
    }
    System.out.println("El tiempo fue de "+ (tiempoFinal-tiempoInicio) +
                       " milisegundos con " + idMaximo + " de cliente ");
}
/**
* Este es el metodo central que se encarga de recivir los parametros
* dados por el usuario, y en caso de ser necesario desplegar una pequena
* ayuda. Cuando los datos ingresados son correctos crea, inicializa y lanza
* al CoordinadorIntegral para que este realice su trabajo.
*/
public static void main(String Args[]){
    if (Args.length < 5){
        System.out.println("Modo de Uso: Java Coordinador <idMinimo>" +
                           " <idMaximo> <Numero Interacciones>" +
                           " <EjeX> <Ejey>");
        System.exit(0);
    }
    CoordinadorSORDirecto xx = new CoordinadorSORDirecto(Byte.parseByte(Args[0]),
                                                       Byte.parseByte(Args[1]),
                                                       Integer.parseInt(Args[2]),
                                                       Integer.parseInt(Args[3]),
                                                       Integer.parseInt(Args[4]));
    try {
        xx.crearConexion();
    } catch (Exception e){
```

```
            e.printStackTrace();
            System.exit(1);

        }
        xx.atender();
    }

}
```

B.6 Cliente de la Integración Numérica

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.awt.Container;
import java.awt.FlowLayout;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
```

```
/**
 * Esta clase es el cliente de integracion, y como tal comprende la parte visual
 * y la parte interna logica de calculo. La seccion interna de logica de calculo
 * se encuentra dada por una clase interna de calculo que en este caso se llama
 * "Integrador".
 */
public class ClienteIntegral extends JApplet {
    private static final long serialVersionUID = 1L; //serial default

    private Socket conexion;
    private DataInputStream entradaDatos;
    private DataOutputStream salidaDatos;
    private String host;
    private int port = 4545;
    private byte id;

    private byte idDevuelta;
    private byte comando;

    //para la parte visual
    private JButton comenzar;
    private JLabel estadoApplet;

/**
 * Este es el metodo init del proceso de inicio de un JApplet,
 * en este caso aqui se inicializan todos los elementos relacionados
 * al GUI y algunos elementos de conexion como son el host name.
 * Tambien inicia el proceso de conexion y comunicacion con el
 * coordinador. (esto significa que no requiere de accion extra
 * por parte del usuario para inciarse)
 */
    public void init(){
        comenzar = new JButton("Comenzar");
        //comenzar.addActionListener(this);
        estadoApplet = new JLabel("Cliente Integral");

        Container cc = getContentPane();
        cc.setLayout(new FlowLayout());
        cc.add(estadoApplet);
        cc.add(comenzar);

        host = getCodeBase().getHost();
        crearConexion();
        Integrador mul = new Integrador();
        mul.start();

    }
    /**
     * Este metodo crea la conexion (socket) hacia el servidor
     * que en este caso es el broker. Ademas de eso inicializa
     * los canales de comunicacion DataInputStream y DataOutputStream.
     */
    public void crearConexion(){
        try {
            conexion = new Socket(host, port);
```

```
            entradaDatos = new DataInputStream(conexion.getInputStream());
            salidaDatos = new DataOutputStream(conexion.getOutputStream());
            id = entradaDatos.readByte();

        } catch (Exception e){
            e.printStackTrace();
        }
    }
    /**
     * Esta clase esta encargada de realizar las peticiones de datos y de
     * realizar los calculos necesarios para obtener los resultados que luego
     * los envia de regreso.
     */
    public class Integrador extends Thread {
private double inicio;
        private double fin;
        private double delta;
private double resultado = 0;
/**
 * Este metodo se encarga de enviar el comando de peticion de datos
 * al coordinador. El cual si tiene datos para entregar los devolvera
 * luego de esta peticion.
 */
private void envioComandoPeticionDatos() throws Exception {
salidaDatos.writeByte((byte) 0);
salidaDatos.writeInt(1); //un byte
salidaDatos.writeByte((byte) 1);
}

/**
 * Este metodo recive los datos pedidos por el cliente integrador
 * y los almacena localmente, en caso que el coordinador no tenga
 * mas pedazos para entregar le indica por medio de un mensaje de
 * tamano cero. Este metodo retorna verdadero (true) si la recepcion
 * fue validad y falso (false) si no hay mas datos.
 */
private boolean recepcionDatosPeticion() throws Exception{
entradaDatos.readByte(); //leemos el id, debe ser 0.
//verificamos que el mensaje tenga contenido
if (entradaDatos.readInt() > 0) {
inicio = entradaDatos.readDouble();
                        fin = entradaDatos.readDouble();
                delta  = entradaDatos.readDouble();
return true;
} else {
//en caso que no tenga contenido no hay mas
//pedazos y terminamos la ejecucion
return false;
}
}

/**
 * Este metodo se encarga de realizar el calculo de la integral, toma
        * como parametros tres doubles <inicio> <fin> y <delta>, siendo inicio
 * el punto de partida de la integral, el fin hasta donde llega y el delta
 * es el tamano de bloque por el que se integra.
```

```
 */
private double CalculoIntegral(double inicio, double fin, double delta) {
//borro el valor del resultado anterior.
resultado = 0;

for (double i = inicio;  i < fin; i+=delta) {
double f = i;
                                resultado += f*delta;
}
return resultado;
}
/**
 * Este metodo envia el resultado calculado (double) de regreso al coordinador
 * toma como parametros un double que es el resultado.
 */
private void envioResultadoCalculo(double resultado) throws Exception {
salidaDatos.writeByte((byte) 0);
salidaDatos.writeInt(9);
                salidaDatos.writeByte(2);
                salidaDatos.writeDouble(resultado);
}
/**
 * Este metodo se encarga de terminar la conexion, cerrando los
 * streams y el socket.
 */
private void cerrarConexcion() throws Exception{
entradaDatos.close();
                salidaDatos.close();
                conexion.close();
}

/**
 * Este metodo es heredado de la clase Thread y es el que ejecuta
 * cuando se llama al metodo start. Aqui se encuentran todas las
 * llamadas necesarias para que el cliente realice su trabajo de calculo.
 */
        public void run() {
            try {
                while(true) {
                    envioComandoPeticionDatos();
    if (recepcionDatosPeticion()) {
envioResultadoCalculo(CalculoIntegral(inicio,fin,delta));
                    }
                    else {
                        cerrarConexcion();
                        break;
                    }
                }
            }
            catch (Exception e){
                //e.printStackTrace();
            }
        }
    }

}
```

B.7 Cliente de la Multiplicación de Matrices

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.awt.Container;
import java.awt.FlowLayout;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;

public class ClienteMatriz extends JApplet {
    private static final long serialVersionUID = 1L; //serial por default
    private double[] lineaA;
    private double[] lineaB;
    private int tamano;
    private int numeroDeBytesPorDoble = 8;

    private Socket conexion;
    private DataInputStream entradaDatos;
```

```
private DataOutputStream salidaDatos;
private String host;
private int port = 4545;
private byte id;

private byte idDevuelta;
private byte comando;

//para la parte visual
private JButton comenzar;
private JLabel estadoApplet;

/**
 * Este metodo se encarga de la inicializacion de elementos
 * de GUI, y ademas de ello llama a la creacion de conexion
 * y luego inicia el proceso interno de calculo sin intervencion
 * del usuario.
 */
public void init(){
    comenzar = new JButton("Comenzar");
    //comenzar.addActionListener(this);
    estadoApplet = new JLabel("Cliente Matriz");

    Container cc = getContentPane();
    cc.setLayout(new FlowLayout());
    cc.add(estadoApplet);
    cc.add(comenzar);

    host = getCodeBase().getHost();
    crearConexion();
    Multiplicador mul = new Multiplicador();
    mul.start();

}
/**
 * Este metodo crea la conexion (socket) hacia el servidor
 * que en este caso es el broker. Ademas de eso inicializa
 * los canales de comunicacion DataInputStream y DataOutputStream.
 */
public void crearConexion(){
    try {
        conexion = new Socket(host, port);
        entradaDatos = new DataInputStream(conexion.getInputStream());
        salidaDatos = new DataOutputStream(conexion.getOutputStream());
        id = entradaDatos.readByte();
        // pido configuracion
        salidaDatos.writeByte((byte) 0);
        salidaDatos.writeInt(1);
        salidaDatos.writeByte((byte) 0);
        // recibo configuracion
        entradaDatos.readByte();
        entradaDatos.readInt();
        tamano = entradaDatos.readInt();
        // System.out.println("el id "+id+" y los tamanos "+tamanoA+" "+tamanoB);

    } catch (Exception e){
```

```
            e.printStackTrace();
        }
    }
    /**
     * Esta clase esta encargada de realizar las peticiones de datos y de
     * realizar los calculos necesarios para obtener los resultados que luego
     * los envia de regreso.
     */
    public class Multiplicador extends Thread {
        int direccionA;
        int direccionB;
double resultado = 0;
/**
 * Este metodo se encarga de enviar el comando de peticion de datos
 * al coordinador. El cual si tiene datos para entregar los devolvera
 * luego de esta peticiion.
 */
private void envioComandoPeticionDatos() throws Exception {
salidaDatos.writeByte((byte) 0);
salidaDatos.writeInt(1); //un byte
salidaDatos.writeByte((byte) 1);
}
/**
 * Este metodo envia el resultado calculado (double) de regreso al coordinador
 * no toma ningun parametro ya que todos son internos.
 */
private void envioResultadoCalculo() throws Exception {
salidaDatos.writeByte((byte) 0);
                salidaDatos.writeInt(17); /*numero de bytes a enviar (2 enteros (4bytes) +
                                                    1 double (8bytes) y 1 bytes de comando) */
                salidaDatos.writeByte(2);
                salidaDatos.writeInt(direccionA);
                salidaDatos.writeInt(direccionB);
                salidaDatos.writeDouble(resultado);
}

/**
 * Este metodo se encarga de realizar el calculo de multiplicacion
         * de matrices, no recive ningun parametro de entrada y como resultado
 * devuelve el resultado como un double.
 */
private double calculoMatricial() {
//borro el valor del resultado anterior.
resultado = 0;
                for(int i = 0; i < tamano;i++) {
                 resultado += lineaA[i]*lineaB[i];
                }
return resultado;
}

/**
 * Este metodo recive los datos pedidos por el cliente
 * y los almacena localmente, en caso que el coordinador no tenga
 * mas pedazos para entregar le indica por medio de un mensaje de
 * tamano cero. Este metodo retorna verdadero (true) si la recepcion
 * fue validad y falso (false) si no hay mas datos.
```

```
 */
private boolean recepcionDatosPeticion() throws Exception{
entradaDatos.readByte(); //leemos el id, debe ser 0.
//verificamos que el mensaje tenga contenido
if (entradaDatos.readInt() > 0) {
direccionA = entradaDatos.readInt();
                        direccionB = entradaDatos.readInt();
                        for (int i = 0; i < tamano; i++){
                            lineaA[i] = entradaDatos.readDouble();
                        }
                        for (int i = 0; i < tamano; i++){
                            lineaB[i] = entradaDatos.readDouble();
                        }
return true;
} else {
//en caso que no tenga contenido no hay mas
//pedazos y terminamos la ejecucion
return false;
}
}

/**
 * Este metodo se encarga de terminar la conexion, cerrando los
 * streams y el socket.
 */
private void cerrarConexcion() throws Exception{
entradaDatos.close();
                salidaDatos.close();
                conexion.close();
}
/**
 * Este metodo es heredado de la clase Thread y es el que ejecuta
 * cuando se llama al metodo start. Aqui se encuentran todas las
 * llamadas necesarias para que el cliente realice su trabajo de calculo.
 */
        public void run(){
            lineaA = new double[tamano];
            lineaB = new double[tamano];
    resultado = 0;
            try {
                while(true) {
                    envioComandoPeticionDatos();
                    if (recepcionDatosPeticion()) {
calculoMatricial();
envioResultadoCalculo();
                    }
                    else {
cerrarConexcion();
                        break;
                    }
                }
            }
            catch (Exception e){
                //e.printStackTrace();
            }
        }
```

```
    }

}
```

B.8 Cliente de SOR con Comunicación Indirecta

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;

/**
 * Este Clase es el cliente para el calculo de SOR para cuando
```

```
 * tenemos comunicacion por medio del broker.
 */
public class ClienteSORBroker extends JApplet implements ActionListener {

    private static final long serialVersionUID = 1L;
    //para identificacion
    private byte id;
    private byte idMinimo;
    private byte idMaximo;

    //datos para el calculo
    private int ejeX;
    private int ejeY;
    private double[][] pedazo;
    private int numeroInteracciones;
    private CalcularSOR calculos;
    private byte temporalID = 0;

    //para conexion
    private Socket conexion;
    private DataInputStream entradaDatos;
    private DataOutputStream salidaDatos;
    private String host;
    private int port = 4545;
    private int numeroDeBytesPorDouble = 8;
    private byte reciviendoDe;
    private int numeroDeBytesALeer;
    private double[] temporal;
    private boolean existeTemporal = false;
    private boolean temporalSuperior = false;

    //para la parte visual
    private JButton comenzar;
    private JLabel estadoApplet;

    //prueba de tiempo
    private long inicioT;
    private long terminaT;

    /**
     * Este metodo se encarga de la inicializacion de elementos
     * de GUI, y ademas de ello llama a la creacion de conexion y servidores
     * y luego inicia el proceso interno de calculo sin intervencion
     * del usuario.
     */
    public void init(){
        //inicio los elementos visuales del applet
        comenzar = new JButton("Comenzar");
        comenzar.addActionListener(this);
        estadoApplet = new JLabel("Cliente SOR");

        Container cc = getContentPane();
        cc.setLayout(new FlowLayout());
        cc.add(estadoApplet);
        cc.add(comenzar);
```

```
        //inicializa el applet
        host = getCodeBase().getHost();
        //host = "localhost";
        try {
            crearConexion();
        } catch (Exception e){
            e.printStackTrace();
        }
        calculos = new CalcularSOR();
        calculos.start();
        comenzar.setEnabled(false);

    }

/**
     * Este metodo crea la conexion (socket) hacia el servidor
     * que en este caso es el broker. Ademas de eso inicializa
     * los canales de comunicacion DataInputStream y DataOutputStream.
     */
    public void crearConexion() throws Exception{
        conexion = new Socket(host, port);
        entradaDatos = new DataInputStream(conexion.getInputStream());
        salidaDatos = new DataOutputStream(conexion.getOutputStream());
        id = entradaDatos.readByte();
        //System.out.println("Recivido Id "+id);

    }
/**
 * Este metodo se encarga de terminar la conexion, cerrando los
 * streams y el socket.
 */
    public void cerrarConexion() throws Exception {
        entradaDatos.close();
        salidaDatos.close();
        conexion.close();
    }
/**
     * Este metodo captura las acciones del boton de la GUI
     * y realiza la accion de lanzar la aplicacion de calculo.
     */
    public void actionPerformed(ActionEvent evento){
        if (evento.getSource() == comenzar){
            estadoApplet.setText("Cliente SOR "+id);
            calculos.start();
            comenzar.setEnabled(false);
        }
    }
/**
     * Esta clase se encarga de realizar todos los procesos
     * necesarios para realizar el calculo, estos comprende desde
     * la iniciacion del servidor interno hasta el intercambio de
     * mensajes entre los vecinos.
 * En este caso con comunicacion Indirecta.
     */
    public class CalcularSOR extends Thread {
/**
```

```
* Este metodo calcula su pedazo de matriz, en este caso
* el calculo es el promedio de los vecinos.
*/
       private void calculoPedazo(){
           for (int i = 1; i < ejeY-1; i++){
               for(int j = 1; j < ejeX-1; j++) {
                   //System.out.println(" "+i+" "+j);
                   pedazo[i][j] = (pedazo[i-1][j] + pedazo[i][j+1] +
                                   pedazo[i+1][j] + pedazo[i][j-1])/4;
               }
           }
       }
           /**
* Este metodo envia los extremos a sus respectivos vecinos
* tomando en cuenta si es que tiene un vecino al que enviar.
*/
       private void envioExtremos() throws Exception {
           if (id != idMinimo) {
               salidaDatos.writeByte(id - (byte) 1);
               salidaDatos.writeInt(ejeX * numeroDeBytesPorDouble);
               for(int i = 0; i < ejeX; i++){
                   salidaDatos.writeDouble(pedazo[1][i]);
               }
           }
           if (id != idMaximo) {
               salidaDatos.writeByte(id + (byte) 1);
               salidaDatos.writeInt(ejeX * numeroDeBytesPorDouble);
               for(int i = 0; i < ejeX; i++){
                   salidaDatos.writeDouble(pedazo[ejeY-2][i]);
               }
           }
       }
          /**
* Este metodo recive los extremos provenientes de los
* vecinos que le corresponden.
*/
       private void recivoExtremos() throws Exception {
           boolean superiorRecivida = false;
           boolean inferiorRecivida = false;

           if (id == idMinimo)
               inferiorRecivida = true;
           if (id == idMaximo)
               superiorRecivida = true;

           if (existeTemporal) {
               // System.out.println("TEMPORAL " + temporalID);
               if (id - (byte) 1 == temporalID){
                   for (int i = 0; i < ejeX; i++){
                       pedazo[0][i] = temporal[i];
                   }
                   existeTemporal = false;
                   inferiorRecivida = true;
               }
               else if (id + (byte) 1 == temporalID){
                   for(int i = 0; i < ejeX; i++){
```

```
                    pedazo[ejeY-1][i] = temporal[i];
                }
                existeTemporal = false;
                superiorRecivida = true;
            }
        }
        while (superiorRecivida == false || inferiorRecivida == false) {
            reciviendoDe = entradaDatos.readByte();
            numeroDeBytesALeer = entradaDatos.readInt();
            if (reciviendoDe == id - (byte) 1) {
                if (inferiorRecivida == false) {
                    for (int i = 0; i < ejeX; i++){
                        pedazo[0][i] = entradaDatos.readDouble();
                    }
                    inferiorRecivida = true;
                }
                else {
                    temporalID = (byte) (id - (byte) 1);
                    if (existeTemporal)
                        System.out.println("ERROR temporal");
                    existeTemporal = true;
                    for (int i = 0; i < ejeX; i++){
                        temporal[i] = entradaDatos.readDouble();
                    }
                }
            }
            if (reciviendoDe == id + (byte) 1) {
                if (superiorRecivida == false) {
                    for (int i = 0; i < ejeX; i++){
                        pedazo[ejeY-1][i] = entradaDatos.readDouble();
                    }
                    superiorRecivida = true;
                }
                else {
                    temporalID = (byte) (id + (byte) 1);
                    if (existeTemporal)
                        System.out.println("ERROR temporal");
                    existeTemporal = true;
                    for (int i = 0; i < ejeX; i++){
                        temporal[i] = entradaDatos.readDouble();
                    }
                }
            }
        }
    }
/**
 * Este metodo envia los resultados luego de terminados
 * todas las interacciones requeridas. Los resultados
 * son enviados el coordinados para que este los almacene.
 */
    private void enviarResultados() throws Exception {
        int inicio = (id == idMinimo ? 0 : 1);
        int finale = (id == idMaximo ? ejeY : ejeY - 1);

        //System.out.println("DEBUG " + finale + " " + inicio + " ejeX " + ejeX);
        int numeroENVIO = (ejeX*(finale - inicio)*numeroDeBytesPorDouble+1);
```

```
            //System.out.println("Cenvia " + numeroENVIO);

            salidaDatos.writeByte(0); //el id del coordinador
            salidaDatos.writeInt(numeroENVIO);
            salidaDatos.writeByte(2);
            for (int i = inicio; i < finale; i++)
                for (int j = 0; j <ejeX; j++)
                    salidaDatos.writeDouble(pedazo[i][j]);
        }
/**
     * Este metodo realiza la peticion de datos para la configuracion
     * recoje estos datos (matriz pedazo que le toca).
     */
private void recepcionDatosConfiguracion() throws Exception {
byte source;
                //ahora tengo que recivir los demas datos de configuracion
                //primero los datos provenientes del coordinador

                sleep(5000);

                salidaDatos.writeByte(0); // le escribo al coordinador
                salidaDatos.writeInt(1); //numero de bytes a enviar
                salidaDatos.writeByte(1); //deseo la configuracion

                source = entradaDatos.readByte();
                while (source != 0) {
                    int size = entradaDatos.readInt()/8;
                    temporalID = source;
                    if (existeTemporal)
                        System.out.println("ERROR temporal inicio " + source);
                    existeTemporal = true;
                    temporal = new double[size];
                    for (int i = 0; i < size; i++)
                        temporal[i] = entradaDatos.readDouble();
                    source = entradaDatos.readByte();
                }

                int n = entradaDatos.readInt(); //numero de bytes a leer
                //System.out.println("RC" + n);
                idMaximo = entradaDatos.readByte(); //cuantos son elementos
                idMinimo = entradaDatos.readByte(); //desde donde se comienza a contar
                ejeX = entradaDatos.readInt();        //numero de elementos por fila
                ejeY = entradaDatos.readInt(); //numero de filas que recive este cliente
                numeroInteracciones = entradaDatos.readInt(); /* numero de interacciones
                                                     que debe realizar este cliente */

                //creamos el pedazo y recivimos los elementos
                pedazo = new double[ejeY][ejeX];
                temporal = new double[ejeX];

                for (int i = 0; i < ejeY; i++){
                    for (int j = 0; j <ejeX; j++){
                        pedazo[i][j] = entradaDatos.readDouble();
                    }
                }
}
```

```
/**
 * Este metodo es heredado de la clase Thread y es el que ejecuta
 * cuando se llama al metodo start. Aqui se encuentran todas las
 * llamadas necesarias para que el cliente realice su trabajo de calculo.
 */
        public void run() {
            try {
recepcionDatosConfiguracion();
int corridas;
                for (corridas = 1; corridas < numeroInteracciones; corridas++) {
                    calculoPedazo();
                    envioExtremos();
                    recivoExtremos();
                }
calculoPedazo();
                enviarResultados();
                cerrarConexion();
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

B.9 Cliente de SOR con Comunicación Directa

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
```

```
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;

/**
 * Esta clase es el cliente para la ejecucion de aplicacion SOR
 * con comunicacion directa entre las partes.
 */

public class ClienteSORDirecto extends JApplet implements ActionListener{

    private static final long serialVersionUID = 1L;
    //para identificarse
    private byte id;
    private byte idMaximo;
    private byte idMinimo;
    private int numeroDeBytesPorDouble = 8;

    //datos para el calculo
    private int ejeX;
    private int ejeY;
    private double[][] pedazo;
    private int numeroInteracciones;

    //para conexion
    private Socket[] conexion;
    private DataInputStream[] entradaDatos;
    private DataOutputStream[] salidaDatos;
    private String host;
    private int port = 4545;

    //datos para la recepcion
    private ServerSocket conexionEntrada;

    //para el thread de calculo
    private CalculoSORDirecto calculos;
```

```
//para la parte visual
private JButton comenzar;
private JLabel estadoApplet;

//para obtner parametros y luego devolver resultados
private String home;
private Socket sHome;
private DataInputStream lecturaParametros;
private DataOutputStream escrituraParametros;

//tiempo
private long inicioT;
private long terminaT;

/**
 * Este metodo se encarga de la inicializacion de elementos
 * de GUI, y ademas de ello llama a la creacion de conexion y servidores
 * y luego inicia el proceso interno de calculo sin intervencion
 * del usuario.
 */
public void init(){
    inicioT = System.currentTimeMillis();
    //iniciacion de elementos graficos
    comenzar = new JButton("comenzar");
    comenzar.addActionListener(this);
    comenzar.setEnabled(false);
    estadoApplet = new JLabel("Cliente SOR - Directo");

    Container cc = getContentPane();
    cc.setLayout(new FlowLayout());
    cc.add(estadoApplet);
    cc.add(comenzar);
    calculos = new CalculoSORDirecto();
    //iniciacion de elementos de conexion
    conexion = new Socket[2];
    entradaDatos = new DataInputStream[2];
    salidaDatos = new DataOutputStream[2];
    //para obtener los parametros
    obtencionParametros();
    estadoApplet.setText("Cliente SOR - Directo " + id);

    try {
        calculos.iniciarServer();
        //System.out.println("Termino la Inicializacion");
        calculos.start();
    } catch (Exception e) {
        e.printStackTrace();
    }

}
/**
 * Este metodo realiza la peticion de datos para la configuracion
 * recoje estos datos (matriz pedazo que le toca). Y finalmente
 * obtiene los datos para realizar la conexion directa.
 */
private void obtencionParametros(){
```

```
//por el momento solo es para obtner el id
try {
    home = getCodeBase().getHost();
    sHome  = new Socket(home,port);
    lecturaParametros = new DataInputStream(sHome.getInputStream());
    escrituraParametros = new DataOutputStream(sHome.getOutputStream());

    id = lecturaParametros.readByte(); //(broker)
    //System.out.println("id = " + id);
    //primero enviamos el ip nuestro
    byte[] dir = InetAddress.getLocalHost().getAddress();
    escrituraParametros.writeByte(0); //para el broker
    escrituraParametros.writeInt(5); // 1 byte comando + 4 de dir
    escrituraParametros.writeByte(3); //comando 3
    escrituraParametros.write(dir);

    //segundo pedimos los parametros de inicializacion
    escrituraParametros.writeByte(0); //para el broker
    escrituraParametros.writeInt(1); //un byte de datos
    escrituraParametros.writeByte(1); //comando 1

    lecturaParametros.readByte(); // id del broker
    lecturaParametros.readInt(); //# lecturas
    idMaximo = lecturaParametros.readByte();
    idMinimo = lecturaParametros.readByte();
    ejeX = lecturaParametros.readInt();
    ejeY = lecturaParametros.readInt();
    numeroInteracciones = lecturaParametros.readInt();
    //ahora la recepcion de la matriz
    pedazo = new double[ejeY][ejeX];
    for (int i = 0; i < ejeY; i++){
        for (int j = 0; j < ejeX; j++){
            pedazo[i][j] = lecturaParametros.readDouble();
        }
    }

    //tercero tratamos de conocer la dir que tenemos conectarnos
    if (id != idMaximo) {
        boolean listo = false;
        while (!listo) {
            escrituraParametros.writeByte(0);
            escrituraParametros.writeInt(1);
            escrituraParametros.writeByte(4);
            lecturaParametros.readByte(); //id broker
            if (lecturaParametros.readInt() == 0) {
                wait(10); // no esta listo, esperamos
            } else {
                byte[] IpByte = new byte[4];
                lecturaParametros.readFully(IpByte);
                host = InetAddress.getByAddress(IpByte).getHostAddress();
                listo = true;
            }
        }
    }
} catch (Exception e){
    e.printStackTrace();
```

```
        }
    }

    /**
     * Este metodo captura las acciones del boton de la GUI
     * y realiza la accion de lanzar la aplicacion de calculo.
     */
    public void actionPerformed(ActionEvent evento){
        if (evento.getSource() == comenzar){
            estadoApplet.setText("Cliente SOR " + id);
            calculos.start();
            comenzar.setEnabled(false);
        }
    }
    /**
     * Esta clase se encarga de realizar todos los procesos
     * necesarios para realizar el calculo, estos comprende desde
     * la iniciacion del servidor interno hasta el intercambio de
     * mensajes entre los vecinos.
     */
    public class CalculoSORDirecto extends Thread {
/**
 * Este metodo calcula su pedazo de matriz, en este caso
 * el calculo es el promedio de los vecinos.
 */
        private void calculoPedazo(){
            for (int i = 1; i < ejeY-1; i++){
                for(int j = 1; j < ejeX-1; j++){
                    pedazo[i][j] = (pedazo[i-1][j] + pedazo[i][j+1] +
                                    pedazo[i+1][j] + pedazo[i][j-1])/4;
                }
            }
        }
/**
 * Este metodo inicia el servidor ("SocketServer")
 */
        public void iniciarServer() throws IOException{
            if (id != idMinimo){
                conexionEntrada = new ServerSocket(port);
            }
        }
        /**
 * Este metodo inicia los canales de comunicacion, para lo cual
 * toma en cuanta el id que posee para conocer con quien debe
 * conectarse y de quien debe escuchar conexion.
 */
        private void iniciacionCanalesComunicacion() throws UnknownHostException,
                                                            IOException,
                                                            InterruptedException {

            if (id != idMinimo) {
                //solo tengo que crear el socket Server
                conexion[1] = conexionEntrada.accept();
                entradaDatos[1] = new DataInputStream(conexion[1].getInputStream());
                salidaDatos[1] = new DataOutputStream(conexion[1].getOutputStream());
            }
```

```
        if (id != idMaximo){
            //solo tengo que conectarme con mi siguiente
            boolean conectado = false;
            while (!conectado) {
                try {
                    conexion[0] = new Socket(host,port);
                    if (conexion[0].isConnected()) {
                        entradaDatos[0] = new DataInputStream(conexion[0].getInputStream());
                        salidaDatos[0] = new DataOutputStream(conexion[0].getOutputStream());
                        conectado = true;
                    }
                } catch (Exception e){
                    //fallo la conexion con mi siguiente
                    sleep(10);
                }
            }
        }
    }
    /**
* Este metodo termina con los canales de comunicacion creados
* de forma segura tomando en cuanta cuales fueron en realidad
* creador.
*/
    private void terminacionCanalesComunicacion() throws IOException{
        if (id != idMaximo) {
            //cerramos el canal de comunicaion superior
            entradaDatos[0].close();
            salidaDatos[0].close();
            conexion[0].close();
        }
        if (id != idMinimo) {
            entradaDatos[1].close();
            salidaDatos[1].close();
            conexion[1].close();
        }
        //ahora para todos
        escrituraParametros.close();
        lecturaParametros.close();
        sHome.close();
    }
    /**
* Este metodo envia los extremos a sus respectivos vecinos
* tomando en cuenta si es que tiene un vecino al que enviar.
*/
    private void enviarExtermos() throws IOException {
        if (id != idMaximo){
            //envio la fila inferior
            for (int i = 0; i < ejeX; i++){
                salidaDatos[0].writeDouble(pedazo[ejeY-1][i]);
            }
        }
        if (id != idMinimo) {
            //envio la fila superior
            for(int i = 0; i <ejeX; i++){
                salidaDatos[1].writeDouble(pedazo[1][i]);
```

```
                }
            }
        }
        /**
* Este metodo recive los extremos provenientes de los
* vecinos que le corresponden.
*/
        private void recivirExtremos() throws IOException {
            if (id != idMaximo){
                //recivo la fila inferior
                for(int i = 0; i < ejeX; i++){
                    pedazo[ejeY-1][i] = entradaDatos[0].readDouble();
                }
            }
            if(id != idMinimo){
                //recivo la fila superior
                for(int i = 0; i < ejeX; i++){
                    pedazo[1][i] = entradaDatos[1].readDouble();
                }
            }
        }
        /**
* Este metodo envia los resultados luego de terminados
* todas las interacciones requeridas. Los resultados
* son enviados el coordinados para que este los almacene.
*/
        private void enviarResultados() throws Exception {
            int inicio = (id == idMinimo ? 0 : 1);
            int finale = (id == idMaximo ? ejeY : ejeY - 1);
            int numeroEnvio = (ejeX*(finale - inicio)*numeroDeBytesPorDouble+1);

            escrituraParametros.writeByte(0); //el id del coordinador
            escrituraParametros.writeInt(numeroEnvio);// # bytes
            escrituraParametros.writeByte(2); //comando (2)
            for (int i = inicio; i < finale; i++)
                for (int j = 0; j <ejeX; j++)
                    escrituraParametros.writeDouble(pedazo[i][j]);
        }

        /**
* Este metodo es heredado de la clase Thread y es el que ejecuta
* cuando se llama al metodo start. Aqui se encuentran todas las
* llamadas necesarias para que el cliente realice su trabajo de calculo.
*/
        public void run() {
            try {
                iniciacionCanalesComunicacion();
                for (int i = 1; i < numeroInteracciones; i++){
                    calculoPedazo();
                    enviarExtermos();
                    recivirExtremos();
                    //System.out.println("Terminada la interaccion "+i);
                }
                calculoPedazo();
                enviarResultados();
                terminacionCanalesComunicacion();
```

```
            } catch (Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

Apéndice C

Código Fuente de los *Micro–benchmarks*

C.1 Velocidad de Transferencia — Productor

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.awt.Container;
```

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;

/* Este es el Despachador que se desarrollo para la prueba
 * de velocidad de transferencia (Bandwidth)
 */
public class AppletDespachador  extends JApplet implements ActionListener{
    private String host;
    private int port;
    private int numeroDespachos;
    private int receptor;

    private JLabel estadoApplet;
    private JButton inicioParada;

    private static final long serialVersionUID = 1L;

    public void init() {

        estadoApplet = new JLabel("Empezar -- DESPACHADOR -- BAND");
        inicioParada = new JButton("Empezar");

        host = getCodeBase().getHost();
        port = 4545;
        numeroDespachos = 10000;
        /* en teoria deberia iniciarlizarse primero el receptor
           y por ende ser la posicion 0 */
        receptor = 0;


        Container contenedor = getContentPane();
        contenedor.setLayout(new FlowLayout());
        contenedor.add(estadoApplet);
        inicioParada.addActionListener(this);
        contenedor.add(inicioParada);


    }

    public void actionPerformed(ActionEvent evento){
        if (evento.getSource() == inicioParada){
            HiloReceptor recep = new HiloReceptor();
            recep.start();
        }
    }

    class HiloReceptor extends Thread{
        public int tamano = 1024;
```

```
        public byte[] unKiloByte = new byte[tamano];
        public HiloReceptor() {}
        public long inicio;
        public long termino;

        public void run() {
            try {
                Socket xx = new Socket(host,port);
                xx.setTcpNoDelay(true);
                xx.setSendBufferSize(128*1024);
                xx.setReceiveBufferSize(128*1024);
                DataOutputStream escri = new DataOutputStream(xx.getOutputStream());
                inicio = System.currentTimeMillis();
                for (int i = 0; i < numeroDespachos; i++){
                    escri.writeByte(receptor);
                    escri.writeInt(tamano);
                    escri.write(unKiloByte);
                    //System.out.println("" + i); //codigo de DEBUG
                }
                termino = System.currentTimeMillis();
                escri.close();
                xx.close();
                System.out.println("ancho en milis\t "
                                   + ((numeroDespachos*1024)/(termino-inicio)));

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

C.2 Velocidad de Transferencia — Receptor

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
```

```
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;

/* Este es el Receptor que se desarrollo para la prueba
 * de velocidad de transferencia (Bandwidth)
 */
public class AppletReceptor extends JApplet implements ActionListener{
    private String host;
    private int port;

    private JLabel estadoApplet;
    private JButton inicioParada;

    private static final long serialVersionUID = 1L;

    public void init() {

        estadoApplet = new JLabel("Empezar -- TERMINADOR -- BAND");
        inicioParada = new JButton("Empezar");

        host = getCodeBase().getHost();
        port = 4545;

        Container contenedor = getContentPane();
        contenedor.setLayout(new FlowLayout());
        contenedor.add(estadoApplet);
        inicioParada.addActionListener(this);
        contenedor.add(inicioParada);
    }

    public void actionPerformed(ActionEvent evento){
        if (evento.getSource() == inicioParada){
```

```
            HiloReceptor recep = new HiloReceptor();
            recep.start();
        }
    }

    class HiloReceptor extends Thread{

        public HiloReceptor() {}
        byte[] lecturaUnica = new byte[1024];
        public void run() {
            try {
                Socket xx = new Socket(host,port);
                xx.setTcpNoDelay(true);
                xx.setReceiveBufferSize(128*1024);
                xx.setSendBufferSize(128*1024);
                DataInputStream leer = new DataInputStream(xx.getInputStream());
                DataOutputStream salida = new DataOutputStream(xx.getOutputStream());

                while(xx.isConnected()){
                    leer.read(lecturaUnica);
                }
                salida.close();
                leer.close();
                xx.close();
                //termine

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

C.3 Latencia — Productor

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
```

```
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.nio.ByteBuffer;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;

/* Este es el Despachador que se desarrollo para la prueba
 * de latencia
 */
public class AppletDespachador  extends JApplet implements ActionListener{
    private String host;
    private int port;
    private int numeroDespachos;
    private int receptor;

    private JLabel estadoApplet;
    private JButton inicioParada;

    private static final long serialVersionUID = 1L;

    public void init() {

        estadoApplet = new JLabel("Empezar -- DESPACHADOR -- BAND");
        inicioParada = new JButton("Empezar");

        host = getCodeBase().getHost();
        port = 4545;
        numeroDespachos = 10000;
        /* en teoria deberia iniciarlizarse primero
           el receptor y por ende ser la posicion 0 */
        receptor = 0;
```

```
    Container contenedor = getContentPane();
    contenedor.setLayout(new FlowLayout());
    contenedor.add(estadoApplet);
    inicioParada.addActionListener(this);
    contenedor.add(inicioParada);


}

public void actionPerformed(ActionEvent evento){
    if (evento.getSource() == inicioParada){
        HiloReceptor recep = new HiloReceptor();
        recep.start();
    }
}

class HiloReceptor extends Thread{

    public HiloReceptor() {}
    public long inicio;
    public long termino;

    public void run() {
        try {
            Socket xx = new Socket(host,port);
            xx.setTcpNoDelay(true);
            xx.setSendBufferSize(128*1024);
            xx.setReceiveBufferSize(128*1024);
            DataOutputStream escri = new DataOutputStream(xx.getOutputStream());
            DataInputStream lectura = new DataInputStream(xx.getInputStream());
            byte[] enviar = new byte[5];
            enviar[0] = (byte) 0;

            enviar[1] = (byte) 0;
            enviar[2] = (byte) 0;
            enviar[3] = (byte) 0;
            enviar[4] = (byte) 0;

            inicio = System.currentTimeMillis();
            for (int i = 0; i < numeroDespachos; i++){
                escri.write(enviar);
                lectura.readInt();
                lectura.readByte();
            }
            termino = System.currentTimeMillis();
            escri.close();
            xx.close();
            System.out.println("ancho en milis\t"
                               + ((termino-inicio)/(double)numeroDespachos)
                               + " " + (termino-inicio));

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
}
```

C.4 Latencia — Receptor

```
/*
 * Copyright (C) 2006 Pablo Bustamante.  All rights reserved.
 * Systems Engineering Department, University San Francisco of Quito.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * 3. The name of the authors may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTIONS) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE), PRODUCT LIABILITY, OR OTHERWISE ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */


import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.nio.ByteBuffer;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;

/* Este es el Receptor que se desarrollo para la prueba
 * de latencia
```

```
 */
public class AppletReceptor extends JApplet implements ActionListener{
    private String host;
    private int port;
    private JLabel estadoApplet;
    private JButton inicioParada;

    private static final long serialVersionUID = 1L;

    public void init() {

        estadoApplet = new JLabel("Empezar -- TERMINADOR -- BAND");
        inicioParada = new JButton("Empezar");

        host = getCodeBase().getHost();
        port = 4545;

        Container contenedor = getContentPane();
        contenedor.setLayout(new FlowLayout());
        contenedor.add(estadoApplet);
        inicioParada.addActionListener(this);
        contenedor.add(inicioParada);
    }

    public void actionPerformed(ActionEvent evento){
        if (evento.getSource() == inicioParada){
            HiloReceptor recep = new HiloReceptor();
            recep.start();
        }
    }

    class HiloReceptor extends Thread{

        public HiloReceptor() {}
        public void run() {
            try {
                Socket xx = new Socket(host,port);
                xx.setTcpNoDelay(true);
                xx.setReceiveBufferSize(128*1024);
                xx.setSendBufferSize(128*1024);
                DataInputStream leer = new DataInputStream(xx.getInputStream());
                DataOutputStream salida = new DataOutputStream(xx.getOutputStream());

                byte[] enviar = new byte[5];
                enviar[0] = (byte) 1;
                enviar[1] = (byte) 0;
                enviar[2] = (byte) 0;
                enviar[3] = (byte) 0;
                enviar[4] = (byte) 0;

                boolean tt = true;
                while(tt){
                    leer.readByte();
                    salida.write(enviar);
                }
                salida.close();
```

```
                leer.close();
                xx.close();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```